NASA/CR-1999-209722
ICASE Report No. 99-43

Dependent Types and Explicit Substitutions

César Muñoz
ICASE, Hampton, Virginia

November 1999

NASA/CR-1999-209722
ICASE Report No. 99-43

Dependent Types and Explicit Substitutions

César Muñoz
ICASE, Hampton, Virginia

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA

Operated by Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NAS1-97046

November 1999

DEPENDENT TYPES AND EXPLICIT SUBSTITUTIONS

CÉSAR MUÑOZ*

Abstract. We present a dependent-type system for a λ-calculus with explicit substitutions. In this system, meta-variables, as well as substitutions, are first-class objects. We show that the system enjoys properties like type uniqueness, subject reduction, soundness, confluence and weak normalization.

Key words. explicit substitutions, dependent types, lambda-calculus

Subject classification. Computer Science

1. Introduction. Since the $\lambda\sigma$-calculus of explicit substitutions was introduced in [1], several other variants of explicit substitution calculi have been proposed; among others [38, 27, 20, 4, 28, 7, 24, 31, 10, 33]. By using substitutions as first-class objects, and de Bruijn indices notation for variables, the $\lambda\sigma$-calculus allows a first-order encoding of the λ-calculus. In consequence, technical nuisances due to higher-order aspects of the λ-calculus, for example α-conversion, can be minimized or eliminated in explicit substitution calculi. For instance, higher-order unification problems have been reformulated in a first-order setting via some variants of $\lambda\sigma$ [8, 9, 25, 5].

However, explicit substitutions are not free of difficulties. Typed versions of these calculi lead to unexpected problems. It is well known now that $\lambda\sigma$ does not preserve strong normalization [30], that is, well-typed terms may not terminate in $\lambda\sigma$. Furthermore, as a rewrite system, $\lambda\sigma$ is not confluent on open terms [7].

In constructive logic, explicit substitutions and open terms form a framework to represent *incomplete proofs*, i.e., proofs under development [29, 32]. In this approach, *meta-variables* are place-holders in a proof-term, and an explicit substitution notation is necessary to delay the application of substitutions to meta-variables waiting to be instantiated. Meta-variables have also been used as unification variables in the higher-order unification methods presented in [8, 9, 25].

In order to apply explicit substitution techniques in a dependent-type framework, we develop a λ-calculus of explicit substitutions, called $\lambda\Pi_{\mathcal{L}}$, with dependent types and support for meta-variables.

The rest of this section gives an overview of the dependent-type theory in which we are interested, and to the simply-typed version of $\lambda\sigma$. We finish the section with a discussion about the main difficulties to set the $\lambda\sigma$-calculus in a dependent-type theory. In Section 2 we present the $\lambda\Pi_{\mathcal{L}}$-calculus. Just as the λ-calculus extended with the η-rule, which is not confluent on terms with type annotations (not necessarily well-typed), $\lambda\Pi_{\mathcal{L}}$ is not confluent due to type annotations on substitutions. However, using a technique proposed by Geuvers in [11], we prove that it is confluent on well-typed expressions. We show how to adapt Geuvers' technique to $\lambda\Pi_{\mathcal{L}}$ in Section 3. In Section 4 we show the elementary typing properties of $\lambda\Pi_{\mathcal{L}}$: sort soundness, type uniqueness, subject reduction and soundness. In Section 5 we prove the main properties on well-typed $\lambda\Pi_{\mathcal{L}}$-expressions: weak normalization, Church-Rosser, and confluence. In the last section we discuss related work and summarize our work.

*Institute for Computer Applications in Science and Engineering, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23681-2199, email: munoz@icase.edu. This research was supported by INRIA - Rocquencourt while the author was an international fellow at the INRIA institute, and by the National Aeronautics and Space Administration under NASA Contract NAS1-97046 while he was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-2199.

1.1. Dependent types. The Dependent Type theory, namely $\lambda\Pi$ [18], is a conservative extension of the simply-typed λ-calculus. It allows a finer stratification of terms by generalizing the function space type. In fact, in $\lambda\Pi$, the type of a function $\lambda x{:}A.M$ is $\Pi x{:}A.B$ where B (the type of M) may depend on x. Hence, the type $A \to B$ of the simply-typed λ-calculus is just a notation in $\lambda\Pi$ for the product $\Pi x{:}A.B$ where x does not appear free in B.

From a logical point of view, the $\lambda\Pi$-calculus allows representation of proofs in the first-order intuitionistic logic using universal quantification. Via the types-as-proofs principle, a term of type $\Pi x{:}A.B$ is a proof-term of the proposition $\forall x{:}A.B$.

Terms in $\lambda\Pi$ can be variables $x, y, \ldots$, applications $(M\ N)$, abstractions $\lambda x{:}A.M$, products $\Pi x{:}A.B$, or one of the sorts *Type*, *Kind*.[1] Notice that terms and types belong to the same syntactical category. Thus, $\Pi x{:}A.B$ is a term, as well as $\lambda x{:}A.M$. However, terms are stratified in several levels according to a type discipline. For instance, given an appropriate context of variable declarations, $\lambda x{:}A..M : \Pi x{:}A..B$, $\Pi x{:}A..B$: *Type*, and *Type* : *Kind*. The term *Kind* cannot be typed in any context, but it is necessary since a circular typing as *Type* : *Type* leads to the Girard's paradox [15].

Typing judgments in $\lambda\Pi$ have the form

$$\Gamma \vdash M : A$$

where Γ is a *context* of variable declarations, that is, a set of type assignments for free variables. We use the Greek letters Γ, Δ to range over contexts. Since types may be ill-typed, typing judgments for valid contexts are also necessary. The notation

$$\vdash \Gamma$$

captures that types in Γ are well-typed. The $\lambda\Pi$-type system is given in Fig. 1.1.

In a higher-order logic, as $\lambda\Pi$, it may happen that two syntactically different types become identical via β-conversion. Rule (Conv) uses the equivalence relation $\equiv_\beta$ which is defined as the reflexive and transitive closure of the relation induced by the β-rule: $(\lambda x{:}A.M\ N) \longrightarrow M[N/x]$. We recall that $M[N/x]$ is just a notation for the atomic substitution of the free occurrences of x in M by N, with renaming of bound variables in M when necessary.

1.2. Explicit substitutions and simple types. The $\lambda\sigma$-calculus [1] is a first-order rewrite system with two sorts of expressions: terms and substitutions.

Simple types are generated from a denumerable set of basic types $a, b, \ldots$ and their functional closure, i.e., if A, B are simple types, then $A \to B$ is also a simple type. Well-formed expressions in the simply-typed $\lambda\sigma$-calculus are defined by the following grammar:

Terms	M, N	$::=$	$\underline{1} \mid (M\ N) \mid \lambda_A.M \mid M[S]$
Substitutions	S, T	$::=$	$id \mid \uparrow \mid M \cdot S \mid S \circ T$
Types	A, B	$::=$	$a, b, \ldots \mid A \to B$

In $\lambda\sigma$, free and bound variables are represented by de Bruijn indices. They are encoded by means of the constant $\underline{1}$ and the substitution $\uparrow$. We write $\uparrow^n$ as a shorthand for $\overbrace{\uparrow \circ \ldots \circ \uparrow}^{n\text{-times}}$. We overload the notation $\underline{i}$ to

[1]The names *Type* and *Kind* are not standard, other couples of names used in the literature are: (*Set*, *Type*), (*Prop*, *Type*) and $(*, \Box)$.

$$\frac{}{\vdash \{\}}\ \text{(Empty)} \qquad \frac{\begin{array}{c}\Gamma \vdash A : s\\ s \in \{\mathit{Kind}, \mathit{Type}\}\\ x \text{ is a fresh variable}\end{array}}{\vdash \Gamma \cup \{x : A\}}\ \text{(Var-Decl)}$$

$$\frac{\vdash \Gamma}{\Gamma \vdash \mathit{Type} : \mathit{Kind}}\ \text{(Type)} \qquad \frac{\begin{array}{c}\vdash \Gamma\\ (x : A) \in \Gamma\end{array}}{\Gamma \vdash x : A}\ \text{(Var)}$$

$$\frac{\begin{array}{c}\Gamma \vdash A : \mathit{Type}\\ x \text{ does not appear in } \Gamma\\ \Gamma \cup \{x : A\} \vdash B : s\\ s \in \{\mathit{Kind}, \mathit{Type}\}\end{array}}{\Gamma \vdash \Pi x{:}A.B : s}\ \text{(Prod)} \qquad \frac{\begin{array}{c}\Gamma \vdash A : \mathit{Type}\\ x \text{ does not appear in } \Gamma\\ \Gamma \cup \{x : A\} \vdash M : B\\ \Gamma \cup \{x : A\} \vdash B : s\\ s \in \{\mathit{Kind}, \mathit{Type}\}\end{array}}{\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B}\ \text{(Abs)}$$

$$\frac{\begin{array}{c}\Gamma \vdash M : \Pi x{:}A.B\\ \Gamma \vdash N : A\end{array}}{\Gamma \vdash (M\ N) : A[N/x]}\ \text{(Appl)} \qquad \frac{\begin{array}{c}\Gamma \vdash M : A\\ \Gamma \vdash B : s\\ s \in \{\mathit{Kind}, \mathit{Type}\}\\ A \equiv_\beta B\end{array}}{\Gamma \vdash M : B}\ \text{(Conv)}$$

FIG. 1.1. *The* $\lambda\Pi$-*system*

represent the $\lambda\sigma$-term corresponding to the index i, i.e.,

$$\underline{i} = \begin{cases} \underline{1} & \text{if } i = 1\\ \underline{1}[\uparrow^n] & \text{if } i = n + 1. \end{cases}$$

An explicit substitution denotes a mapping from indices to terms. Thus, id maps each index i to the term $\underline{i}$, $\uparrow$ maps each index i to the term $\underline{i+1}$, $S \circ T$ is the composition of the mapping denoted by T with the mapping denoted by S (notice that the composition of substitution follows a reverse order with respect to the usual notation of function composition), and finally, $M \cdot S$ maps the index 1 to the term M, and recursively, the index $i + 1$ to the term mapped by the substitution S on the index i.

A *context* in $\lambda\sigma$ is a list of types. The empty context is written ϵ. A context with head A and rest Γ is written $A.\Gamma$. In that case, A is the type of the index 1, the head of Γ (if Γ is not empty) is the type of the index 2, and so on.

The type of a substitution is a context. This choice seems natural since substitutions denote mapping from indices to terms, and contexts are list of types. In fact, if the type of a substitution S is the context $A.\Delta$, the type of the term mapped by the substitution S on the index 1 is A, and so for the rest of indices. Typing judgment for substitutions in $\lambda\sigma$ have the form:

$$\Gamma \vdash S \triangleright \Delta.$$

The $\lambda\sigma$-calculus and its typing rules are presented in Fig. 1.2. When meta-variables of terms are considered, an additional typing rule is necessary to state that each meta-variable is typed in a unique

$$\begin{array}{llll}
(\lambda_A.M\ N) & \longrightarrow & M[N \cdot id] & \text{(Beta)} \\
(M\ N)[S] & \longrightarrow & (M[S]\ N[S]) & \text{(Application)} \\
(\lambda_A.M)[S] & \longrightarrow & \lambda_A.M[\underline{1} \cdot (S \circ \uparrow)] & \text{(Lambda)} \\
M[S][T] & \longrightarrow & M[S \circ T] & \text{(Clos)} \\
\underline{1}[M \cdot S] & \longrightarrow & M & \text{(VarCons)} \\
M[id] & \longrightarrow & M & \text{(Id)} \\
(S_1 \circ S_2) \circ T & \longrightarrow & S_1 \circ (S_2 \circ T) & \text{(Ass)} \\
(M \cdot S) \circ T & \longrightarrow & M[T] \cdot (S \circ T) & \text{(Map)} \\
id \circ S & \longrightarrow & S & \text{(Idl)} \\
S \circ id & \longrightarrow & S & \text{(Idr)} \\
\uparrow \circ (M \cdot S) & \longrightarrow & S & \text{(ShiftCons)} \\
\underline{1} \cdot \uparrow & \longrightarrow & id & \text{(VarShift)} \\
\underline{1}[S] \cdot (\uparrow \circ S) & \longrightarrow & S & \text{(SCons)}
\end{array}$$

$$\frac{}{A.\Gamma \vdash \underline{1} : A}\text{(Var)} \qquad \frac{A.\Gamma \vdash M : B}{\Gamma \vdash \lambda_A.M : A \to B}\text{(Abs)}$$

$$\frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash (M\ N) : B}\text{(Appl)} \qquad \frac{\Gamma \vdash S \rhd \Delta \quad \Delta \vdash M : A}{\Gamma \vdash M[S] : A}\text{(Clos)}$$

$$\frac{}{\Gamma \vdash id \rhd \Gamma}\text{(Id)} \qquad \frac{}{A.\Gamma \vdash \uparrow \rhd \Gamma}\text{(Shift)}$$

$$\frac{\Gamma \vdash S \rhd \Delta_1 \quad \Delta_1 \vdash T \rhd \Delta_2}{\Gamma \vdash T \circ S \rhd \Delta_2}\text{(Comp)} \qquad \frac{\Gamma \vdash M : A \quad \Gamma \vdash S \rhd \Delta}{\Gamma \vdash M \cdot S \rhd A.\Delta}\text{(Cons)}$$

FIG. 1.2. *The simply-typed λσ-calculus [1]*

context by a unique type [8]:

$$\frac{}{\Gamma_X \vdash X : A_X}\text{(Meta}_X\text{)}.$$

The simply-typed $\lambda\sigma$-calculus with meta-variables of terms is confluent [38] and weakly normalizing [17, 33].

1.3. Dependent types and explicit substitutions. A dependent-type system for $\lambda\Pi_{\mathcal{L}}$ is not a simple extension of the simply-typed $\lambda\sigma$-calculus. First of all, it is not clear how to type expressions containing meta-variables. Notice that in a dependent-type theory with de Bruijn indices, the order in which variables are declared in a context is important. In fact, in the context $A.\Gamma$, the indices in A are relative to Γ. But, how is the dependence regarding meta-variables?

Even without considering meta-variables, setting $\lambda\sigma$ in a dependent-type theory presents difficulties. Take, for example, the typing rule for simultaneous substitutions, the (Cons)-rule:

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash S \rhd \Delta}{\Gamma \vdash M \cdot S \rhd A.\Delta}\text{(Cons)}.$$

A dependent-typed version of this rule has the form

$$\frac{\Gamma \vdash M : A[S] \quad \Gamma \vdash S \triangleright \Delta \quad \Delta \vdash A : \mathit{Type}}{\Gamma \vdash M \cdot S \triangleright A.\Delta} \text{(Cons}_\Pi\text{)}.$$

First notice that the type given to M in the premises of the rule is $A[S]$ (up to conversion). The application of the substitution S to the type A is necessary to take into account possible dependencies of variables in A with terms in S. Hence, a type inference algorithm should use a higher-order unification procedure to infer the type of $M \cdot S$ which depends on A.

Another drawback of (Cons$_\Pi$) is that it is not sound with respect to the usual typing properties. In particular, a substitution can be typed with two contexts that are not convertible, i.e., types are not unique modulo conversion. For example, consider the context[2]

$$\Gamma = 0{:}\mathit{nat}.\ l{:}(\Pi n{:}\mathit{nat}.(T\ n)).\ T{:}\mathit{nat} \to \mathit{Type}.\ \mathit{nat}{:}\mathit{Type}$$

and the valid typing judgments

(1.1) $$\Gamma \vdash [x := 0 \cdot \mathit{id}] \triangleright x{:}\mathit{nat}.\ \Gamma$$

(1.2) $$\Gamma \vdash (l\ 0) : (T\ x)[x := 0 \cdot \mathit{id}].$$

Since $(T\ x)[x := 0 \cdot \mathit{id}]$ and $(T\ 0)[x := 0 \cdot \mathit{id}]$ are convertible via $\lambda\sigma$, and $(T\ 0)[x := 0 \cdot \mathit{id}]$ is a valid type, we also have:

(1.3) $$\Gamma \vdash (l\ 0) : (T\ 0)[x := 0 \cdot \mathit{id}].$$

Using (Cons$_\Pi$) with (Eq. 1.1) and (Eq. 1.2), we get:

(1.4) $$\Gamma \vdash [y := (l\ 0) \cdot x := 0 \cdot \mathit{id}] \triangleright y{:}(T\ 0).\ x{:}\mathit{nat}.\ \Gamma$$

and with (Eq. 1.1) and (Eq. 1.3):

(1.5) $$\Gamma \vdash [y := (l\ 0) \cdot x := 0 \cdot \mathit{id}] \triangleright y{:}(T\ x).\ x{:}\mathit{nat}.\ \Gamma.$$

However, $(T\ 0)$ and $(T\ x)$ are not convertible, and then, the substitution $[y := (l\ 0) \cdot x := 0 \cdot \mathit{id}]$ has two types, $y{:}(T\ 0).\ x{:}\mathit{nat}.\ \Gamma$ and $y{:}(T\ x).\ x{:}\mathit{nat}.\ \Gamma$, which are not convertible.

To solve these problems, we use type annotations in substitutions, in a similar way as the Church style λ-calculus —as opposed to the Curry style— annotates binder variables in abstractions. The final version of (Cons$_\Pi$) has the form:

$$\frac{\Gamma \vdash M : A[S] \quad \Gamma \vdash S \triangleright \Delta \quad \Delta \vdash A : \mathit{Type}}{\Gamma \vdash M \cdot_A S \triangleright A.\Delta} \text{(Cons}_\Pi\text{)}.$$

Annotations in substitutions act as reminders of types, and they must be introduced and maintained by the calculus of substitutions. In our previous example, substitutions in Eq. 1.4 and Eq. 1.5 should be annotated with different types.

[2]For readability, we use named variables when discussing examples. Nevertheless, as we have said, $\lambda\sigma$ uses a de Bruijn nameless notation of variables.

A different solution proposed by Bloo in [2] is to introduce substitutions in contexts and to deal with these extended contexts via additional typing rules. This approach is similar to type systems with definitions [41, 3], where closures are typeable, but substitutions are not considered as typeable objects. We discuss this approach in the last section.

When we consider annotated substitutions, the system may lose the subject reduction property due to the non-left-linear rule (SCons): $\underline{1}[S] \cdot_A (\uparrow \circ S) \longrightarrow S$. For instance, take the context

$$\Gamma = m{:}(T\ 0) \to nat.\ 0{:}nat.\ l{:}(\Pi n{:}nat.(T\ n)).\ T{:}nat \to Type.\ nat{:}Type$$

and the substitution

$$S = [y := (l\ 0) \cdot_{(T\ 0)} x := 0 \cdot_{nat} id].$$

We verify that the following typing judgments are valid:

$$\Gamma \vdash S \rhd y{:}(T\ 0).\ x{:}nat.\ \Gamma$$

$$\Gamma \vdash \underline{1}[S] \cdot_{(T\ x)} (\uparrow \circ S) \rhd y{:}(T\ x).\ x{:}nat.\ \Gamma.$$

But also, $\underline{1}[S] \cdot_{(T\ x)} (\uparrow \circ S) \xrightarrow{\text{(SCons)}} S$. However, since $(T\ 0)$ and $(T\ x)$ are not convertible, $\Gamma \not\vdash S \rhd y{:}(T\ x).\ x{:}nat.\ \Gamma$. Therefore, the type of $\underline{1}[S] \cdot_{(T\ x)} (\uparrow \circ S)$ is not preserved by rule (SCons).

The problem here is not the type system but the substitution calculus. Non-left-linear rules —like (SCons)— are not only harmful for typing, but are also usually responsible for non-confluence problems [26, 7].

Nadathur [35] has remarked that in $\lambda\sigma$ with meta-variables of terms, but without meta-variables of substitutions, rule (SCons) is admissible when the following scheme of rule is added to the system: $\underline{1}[\uparrow^n] \cdot \uparrow^{n+1} \longrightarrow \uparrow^n$. Since $\uparrow^n$ is a shorthand, an infinite set of rules is represented by this scheme. Following Nadathur's idea, we present in [33] a variant of $\lambda\sigma$, namely $\lambda_{\mathcal{L}}$, which has the same general features as $\lambda\sigma$, i.e., simple, finite, and first-order presentation, but without rule (SCons) of $\lambda\sigma$.

In this paper, we propose the $\lambda\Pi_{\mathcal{L}}$-calculus, which is based on $\lambda_{\mathcal{L}}$, and show that $\lambda\Pi_{\mathcal{L}}$ is a suitable calculus for our purpose: explicit substitutions, dependent types and support for meta-variables.

2. $\lambda\Pi_{\mathcal{L}}$-Calculus.

As usual in explicit substitution calculi, expressions of $\lambda\Pi_{\mathcal{L}}$ are structured in *terms* and *substitutions.* Since we use the left-linear variant of $\lambda\sigma$, the $\lambda_{\mathcal{L}}$-calculus, we add the sort of *natural numbers.* The $\lambda\Pi_{\mathcal{L}}$-calculus admits meta-variables only on the sort of terms.

The set of well-formed expressions in $\lambda\Pi_{\mathcal{L}}$ is defined by the following grammar:

Natural numbers	n	$::=$	$0 \mid n+1$
Meta-variables	χ	$::=$	$X \mid Y \mid \ldots$
Terms	A, B, M, N	$::=$	$Kind \mid Type \mid \underline{1} \mid \Pi_A.B \mid \lambda_A.M \mid (M\ N) \mid$
			$M[S] \mid \chi$
Substitutions	S, T	$::=$	$\uparrow^n \mid M \cdot_A S \mid S \circ T$

The equivalence relation $\equiv_{\lambda\Pi_{\mathcal{L}}}$ is defined as the symmetric and transitive closure of the relation induced by the rewrite system in Fig. 2.1.

The system $\Pi_{\mathcal{L}}$ is obtained by dropping rule (Beta) from $\lambda\Pi_{\mathcal{L}}$. As shown by Zantema [47], the $\Pi_{\mathcal{L}}$-calculus is strongly normalizing.

$(\lambda_A.M\ N)$	$\longrightarrow$	$M[N \cdot_A \uparrow^0]$	(Beta)
$(\lambda_A.M)[S]$	$\longrightarrow$	$\lambda_{A[S]}.M[\underline{1} \cdot_A (S \circ \uparrow^1)]$	(Lambda)
$(\Pi_A.B)[S]$	$\longrightarrow$	$\Pi_{A[S]}.B[\underline{1} \cdot_A (S \circ \uparrow^1)]$	(Pi)
$(M\ N)[S]$	$\longrightarrow$	$(M[S]\ N[S])$	(Application)
$M[S][T]$	$\longrightarrow$	$M[S \circ T]$	(Clos)
$\underline{1}[M \cdot_A S]$	$\longrightarrow$	M	(VarCons)
$M[\uparrow^0]$	$\longrightarrow$	M	(Id)
$(M \cdot_A S) \circ T$	$\longrightarrow$	$M[T] \cdot_A (S \circ T)$	(Map)
$\uparrow^0 \circ S$	$\longrightarrow$	S	(IdS)
$\uparrow^{n+1} \circ (M \cdot_A S)$	$\longrightarrow$	$\uparrow^n \circ S$	(ShiftCons)
$\uparrow^{n+1} \circ \uparrow^m$	$\longrightarrow$	$\uparrow^n \circ \uparrow^{m+1}$	(ShiftShift)
$\underline{1} \cdot_A \uparrow^1$	$\longrightarrow$	$\uparrow^0$	(Shift0)
$\underline{1}[\uparrow^n] \cdot_A \uparrow^{n+1}$	$\longrightarrow$	$\uparrow^n$	(ShiftS)
$\mathit{Type}[S]$	$\longrightarrow$	Type	(Type)

FIG. 2.1. *The $\lambda\Pi_{\mathcal{L}}$-rewrite system*

LEMMA 2.1. *The $\Pi_{\mathcal{L}}$-calculus is terminating.*

Proof. See [34]. The proof uses the semantic labeling technique [46]. □

The $\lambda\Pi_{\mathcal{L}}$-calculus, just as $\lambda\sigma$, uses the composition operation to achieve confluence on terms with meta-variables. Rules (Idr) and (Ass) of $\lambda\sigma$ are not necessary in $\lambda\Pi_{\mathcal{L}}$.

We adopt the notation $\underline{i}$ as a shorthand for $\underline{1}[\uparrow^n]$ for $i = n+1$. In contrast to $\lambda\sigma$, $\uparrow^n$ is not a shorthand but an explicit substitution in $\lambda\Pi_{\mathcal{L}}$. Indeed, $\uparrow^0$ replaces *id* and $\uparrow^1$ replaces $\uparrow$. In general, $\uparrow^n$ denotes the mapping of each index i to the term $\underline{i+n}$. Using $\uparrow^n$, the scheme of rule proposed by Nadathur can be encoded in a first-order rewrite system. Notice that we do not assume any meta-theoretical property on natural numbers. They are constructed with 0 and $n+1$. Arithmetic calculations on indices are embedded in the rewrite system.

2.1. Meta-variables in $\lambda\Pi_{\mathcal{L}}$.

As we have said, meta-variables are first-class objects in $\lambda\Pi_{\mathcal{L}}$. Just as variables, they have to be declared in order to keep track of possible dependencies between terms and types.

A meta-variable declaration has the form $(X{:}_\Gamma A)$, where Γ and A are, respectively, a context and a type assigned to the meta-variable X. The pair (Γ, A) is unique (modulo $\equiv_{\lambda\Pi_{\mathcal{L}}}$) for each meta-variable. This requirement is enforced by the type system.

A list of meta-variable declarations is called a *signature*. We use the Greek letter Σ to range over signatures. The empty signature is written ϵ. A signature with head $(X{:}_\Gamma A)$ and rest Σ is written $(X{:}_\Gamma A).\ \Sigma$. We overload the notation $\Sigma_1.\ \Sigma_2$ to write the concatenation of the signatures Σ_1 and Σ_2.

The order of the meta-variable declarations is important. In a signature $(X_1{:}_{\Gamma_1} A_1). \ldots (X_n{:}_{\Gamma_n} A_n)$, the type A_i and the context Γ_i, $0 < i \leq n$, may depend only on meta-variables X_j, $i < j \leq n$. The indices in A_i are relative to the context Γ_i.

The main operation on meta-variables is *instantiation*. The instantiation of a meta-variable X with a term M in an expression y (where y is a term or a substitution), denoted by $y\{X \mapsto M\}$, replaces all the occurrences of X in y by M. Application of an instantiation to a context Γ (signature Σ) is denoted by $\Gamma\{X \mapsto M\}$ ($\Sigma\{X \mapsto M\}$). It is defined in the obvious way.

In contrast to substitutions of variables, instantiations of meta-variables allow capturing of variables. Instantiations are not first-class objects, i.e., the application of an instantiation is atomic and external to the $\lambda\Pi_{\mathcal{L}}$-calculus.

2.2. The $\lambda\Pi_{\mathcal{L}}$-type system. In $\lambda\Pi_{\mathcal{L}}$, we consider typing assertions having one of the following forms:

$$\vdash \Sigma; \Gamma$$

to capture that the context Γ is valid in the signature Σ,

$$\Sigma; \Gamma \vdash M : A$$

to capture that the term M has type A (the type M has the kind A) in $\Sigma; \Gamma$, and

$$\Sigma; \Gamma \vdash S \rhd \Delta$$

to capture that the substitution S has the context type Δ in $\Sigma; \Gamma$.

The scoping rules for variables and meta-variables in the above type assertions are as follows. Contexts Γ, Δ, and expressions M, A, S may depend on any meta-variable declared in the respective signature Σ. Indices in M, A, and S are relative to their respective context Γ.

Typing rules for signatures, contexts, terms, and substitutions are all mutually dependent. They are given in Fig. 2.2.

In the following, we use $\vdash \Sigma$, $\vdash \Gamma$, $\Gamma \vdash M : A$, and $\Gamma \vdash S \rhd \Delta$ as shorthands for $\vdash \Sigma; \epsilon$, $\vdash \epsilon; \Gamma$, $\epsilon; \Gamma \vdash M : A$, and $\epsilon; \Gamma \vdash S \rhd \Delta$, respectively.

Since there are no typing rules for *Kind*, the term *Kind* does not occur as a sub-term of a well-typed expression.

The $\lambda\Pi_{\mathcal{L}}$-system types at least as many terms as $\lambda\Pi$. In other words, $\lambda\Pi_{\mathcal{L}}$ is a conservative extension of $\lambda\Pi$.

LEMMA 2.2 (Conservative extension). *Let M, A be ground terms in $\lambda\Pi_{\mathcal{L}}$, and Γ a ground context such that M, A, Γ do not contain explicit substitutions, then $\Gamma \vdash M : A$ in $\lambda\Pi_{\mathcal{L}}$ if and only if $\Gamma \vdash M : A$ in $\lambda\Pi$ (modulo de Bruijn indices translation).*

Proof. By induction on the typing derivation. □

The following lemma states the conditions that guarantee the soundness of instantiation of meta-variables in $\lambda\Pi_{\mathcal{L}}$.

LEMMA 2.3 (Instantiation soundness). *Let M be a term such that $\Sigma_1; \Gamma \vdash M : A$, and Σ a signature having the form $\Sigma_2. (X{:}_\Gamma A). \Sigma_1$,*

1. *if $\vdash \Sigma; \Delta$, then $\vdash \Sigma\{X \mapsto M\}; \Delta\{X \mapsto M\}$,*
2. *if $\Sigma; \Delta \vdash N : B$, then*
   $\Sigma\{X \mapsto M\}; \Delta\{X \mapsto M\} \vdash N\{X \mapsto M\} : B\{X \mapsto M\}$, and
3. *if $\Sigma; \Delta_1 \vdash S \rhd \Delta_2$, then $\Sigma\{X \mapsto M\}; \Delta_1\{X \mapsto M\} \vdash S\{X \mapsto M\} \rhd \Delta_2\{X \mapsto M\}$.*

Proof. By induction on the typing derivation. □

2.3. Type annotations. Type annotations in substitutions are introduced with rules (Beta), (Lambda), and (Pi), and then propagated with rule (Map). They can also be eliminated with rules (VarCons), (ShiftCons), and (Shift0). Notice that the type annotation propagated by rule (Map): $(M \cdot_A S) \circ T \longrightarrow M[T] \cdot_A (S \circ T)$ is A, not $A[T]$.

Consider the following example.

$$\frac{}{\vdash \epsilon; \epsilon}\ (\text{Empty})$$

$$\frac{\Sigma; \Gamma \vdash A : s \quad s \in \{\mathit{Kind}, \mathit{Type}\} \quad X \text{ is a fresh meta-variable}}{\vdash (X{:}_\Gamma A).\, \Sigma}\ (\text{Metavar-Decl})$$

$$\frac{\Sigma; \Gamma \vdash A : s \quad s \in \{\mathit{Kind}, \mathit{Type}\}}{\vdash \Sigma; A.\Gamma}\ (\text{Var-Decl})$$

$$\frac{\vdash \Sigma; \Gamma}{\Sigma; \Gamma \vdash \mathit{Type} : \mathit{Kind}}\ (\text{Type})$$

$$\frac{\vdash \Sigma; A.\Gamma}{\Sigma; A.\Gamma \vdash \underline{1} : A[\uparrow^1]}\ (\text{Var})$$

$$\frac{\Sigma; \Gamma \vdash A : \mathit{Type} \quad \Sigma; A.\Gamma \vdash B : s \quad s \in \{\mathit{Kind}, \mathit{Type}\}}{\Sigma; \Gamma \vdash \Pi_A.B : s}\ (\text{Prod})$$

$$\frac{\Sigma; \Gamma \vdash A : \mathit{Type} \quad \Sigma; A.\Gamma \vdash M : B \quad \Sigma; \Gamma \vdash \Pi_A.B : s \quad s \in \{\mathit{Kind}, \mathit{Type}\}}{\Sigma; \Gamma \vdash \lambda_A.M : \Pi_A.B}\ (\text{Abs})$$

$$\frac{\Sigma; \Gamma \vdash M : \Pi_A.B \quad \Sigma; \Gamma \vdash N : A}{\Sigma; \Gamma \vdash (M\ N) : B[N \cdot_A \uparrow^0]}\ (\text{Appl})$$

$$\frac{\Sigma; \Gamma \vdash S \triangleright \Delta \quad \Sigma; \Delta \vdash M : A \quad \Sigma; \Delta \vdash A : s \quad s \in \{\mathit{Kind}, \mathit{Type}\}}{\Sigma; \Gamma \vdash M[S] : A[S]}\ (\text{Clos})$$

$$\frac{\Sigma; \Gamma \vdash S \triangleright \Delta \quad \Sigma; \Delta \vdash A : \mathit{Kind}}{\Sigma; \Gamma \vdash A[S] : \mathit{Kind}}\ (\text{Clos-Kind})$$

$$\frac{\vdash \Sigma; \Gamma \quad (X{:}_\Delta A) \in \Sigma \quad \Delta \equiv_{\lambda\Pi_\mathcal{L}} \Gamma}{\Sigma; \Gamma \vdash X : A}\ (\text{Metavar})$$

$$\frac{\Sigma; \Gamma \vdash M : A \quad \Sigma; \Gamma \vdash B : s \quad s \in \{\mathit{Kind}, \mathit{Type}\} \quad A \equiv_{\lambda\Pi_\mathcal{L}} B}{\Sigma; \Gamma \vdash M : B}\ (\text{Conv})$$

$$\frac{\Sigma; \Gamma \vdash S \triangleright \Delta_1 \quad \vdash \Sigma; \Delta_2 \quad \Delta_1 \equiv_{\lambda\Pi_\mathcal{L}} \Delta_2}{\Sigma; \Gamma \vdash S \triangleright \Delta_2}\ (\text{Conv-Subs})$$

$$\frac{\vdash \Sigma; \Gamma}{\Sigma; \Gamma \vdash \uparrow^0 \triangleright \Gamma}\ (\text{Id})$$

$$\frac{\vdash \Sigma; A.\Gamma \quad \Sigma; \Gamma \vdash \uparrow^n \triangleright \Delta}{\Sigma; A.\Gamma \vdash \uparrow^{n+1} \triangleright \Delta}\ (\text{Shift})$$

$$\frac{\Sigma; \Gamma \vdash S \triangleright \Delta_1 \quad \Sigma; \Delta_1 \vdash T \triangleright \Delta_2}{\Sigma; \Gamma \vdash T \circ S \triangleright \Delta_2}\ (\text{Comp})$$

$$\frac{\Sigma; \Gamma \vdash M : A[S] \quad \Sigma; \Gamma \vdash S \triangleright \Delta \quad \Sigma; \Delta \vdash A : \mathit{Type}}{\Sigma; \Gamma \vdash M \cdot_A S \triangleright A.\Delta}\ (\text{Cons})$$

FIG. 2.2. *The $\lambda\Pi_\mathcal{L}$-type system*

Let $\Gamma = z{:}nat.\ T{:}nat \to Type.\ nat{:}Type.$ We verify that

$$\Gamma \vdash (\lambda x{:}nat.\lambda f{:}((T\ x) \to nat).\lambda y{:}(T\ x).(f\ y)\ \ z) : ((T\ z) \to nat) \to ((T\ z) \to nat). \tag{2.1}$$

Reducing the (Beta)-redex and distributing the substitution inside the abstraction, we get

$$\begin{array}{ll} (\lambda x{:}nat.\lambda f{:}((T\ x) \to nat).\lambda y{:}(T\ x).(f\ y)\ \ z) & \xrightarrow{\text{(Beta)}} \\ (\lambda f{:}((T\ x) \to nat).\lambda y{:}(T\ x).(f\ y))[x := z \cdot_{nat} \uparrow^0] & \xrightarrow{\Pi_{\mathcal{L}}{}^*} \\ \lambda f{:}((T\ z) \to nat).((\lambda y{:}(T\ x).(f\ y))[f := f \cdot_{(T\ x)\to nat} x := z \cdot_{nat} \uparrow^1]). & \end{array}$$

We will check that the type in Eq. 2.1 is preserved by the reduction.

Thanks to the rewrite rule (Lambda), the type annotation for f in the substitution $[f := f \cdot_{(T\ x)\to nat} x := z \cdot_{nat} \uparrow^1]$ is $(T\ x) \to nat$, that is, the type of the variable f before the distribution of the substitution $[x := z \cdot_{nat} \uparrow^0]$ in the abstraction $\lambda f{:}((T\ x) \to nat).\lambda y{:}(T\ x).(f\ y)$.

The typing rules for substitutions install the right context of variables. For example, the expression $\lambda y{:}(T\ x).(f\ y)$ will be typed in a context where the variable declaration $f : (T\ z) \to nat$ has been replaced by $f : (T\ x) \to nat$. In fact, we verify

$$f{:}(T\ z) \to nat.\ \Gamma \vdash [f := f \cdot_{(T\ x)\to nat} x := z \cdot_{nat} \uparrow^1] \triangleright f{:}(T\ x) \to nat.\ x{:}nat.\ \Gamma \tag{2.2}$$

$$f{:}(T\ x) \to nat.\ x{:}nat.\ \Gamma \vdash \lambda y{:}(T\ x).(f\ y) : (T\ x) \to nat \tag{2.3}$$

hence, by rule (Clos) applied to Eq. 2.2 and Eq. 2.3:

$$f{:}(T\ z) \to nat.\ \Gamma \vdash (\lambda y{:}(T\ x).(f\ y))[f := f \cdot_{(T\ x)\to nat} x := z \cdot_{nat} \uparrow^1] : (T\ z) \to nat \tag{2.4}$$

and by rule (Abs) applied to Eq. 2.4:

$$\begin{array}{ll} \Gamma \vdash & \lambda f{:}((T\ z) \to nat).(\lambda y{:}(T\ x).(f\ y))[f := f \cdot_{(T\ x)\to nat} x := z \cdot_{nat} \uparrow^1] : \\ & ((T\ z) \to nat) \to ((T\ z) \to nat). \end{array}$$

The above example is due to Geuvers and Bloo [13], and it happens to be a counter-example for subject reduction in calculi of explicit substitutions with dependent types where substitutions do not keep track of typing information. The use of annotated substitutions in $\lambda\Pi_{\mathcal{L}}$ keeps the right type when a substitution is propagated under an abstraction or a product. In fact, as we will show below, subject reduction holds in $\lambda\Pi_{\mathcal{L}}$.

However, annotated substitutions raise a technical problem: the $\lambda\Pi_{\mathcal{L}}$-rewrite system is not confluent. The problem even exists if we only consider local confluence on ground terms. In fact, the following critical pair is not joinable in the general case, e.g., assume A and B to be different ground $\lambda\Pi_{\mathcal{L}}$-normal forms:

$$\begin{tikzcd} & (\underline{1} \cdot_A \uparrow^1) \circ (M \cdot_B S) \arrow[dl, "\text{(Shift0);(IdS)}"'] \arrow[dr, "\text{(Map);(VarCons);(ShiftCons);(IdS)}"] & \\ M \cdot_B S & & M \cdot_A S \end{tikzcd}$$

This problem is similar to the one pointed out by Nederpelt for the λ-calculus extended with the η-rule [36]. In that case, the confluence property holds on terms without type annotations in abstractions (λ-calculus in the Curry style), but does not on terms with annotated abstractions (λ-calculus in the Church style). In [11], Geuvers proposes a method to prove confluence for the $\beta\eta$-reduction on well-typed λ-terms written in the Church style. In the next section we adapt this technique in order to prove the confluence property on well-typed $\lambda\Pi_{\mathcal{L}}$ expressions.

$$\begin{array}{lcll}
(\lambda_A.M\ N) & \longrightarrow & M[N\cdot\uparrow^0] & \text{(Beta)}\\
(\lambda_A.M)[S] & \longrightarrow & \lambda_{A[S]}.M[\underline{1}\cdot(S\circ\uparrow^1)] & \text{(Lambda)}\\
(\Pi_A.B)[S] & \longrightarrow & \Pi_{A[S]}.B[\underline{1}\cdot(S\circ\uparrow^1)] & \text{(Pi)}\\
\underline{1}[M\cdot S] & \longrightarrow & M & \text{(VarCons)}\\
(M\cdot S)\circ T & \longrightarrow & M[T]\cdot(S\circ T) & \text{(Map)}\\
\uparrow^{n+1}\circ(M\cdot S) & \longrightarrow & \uparrow^n\circ S & \text{(ShiftCons)}\\
\underline{1}\cdot\uparrow^1 & \longrightarrow & \uparrow^0 & \text{(Shift0)}\\
\underline{1}[\uparrow^n]\cdot\uparrow^{n+1} & \longrightarrow & \uparrow^n & \text{(ShiftS)}
\end{array}$$

FIG. 3.1. *Modified rules in the $\lambda\Pi_{\mathcal{L}}^{\square}$-rewrite system*

3. Geuvers' Lemma. Geuvers' lemma is a weak form of the Church-Rosser property which suffices to prove the main typing properties in systems where confluence on terms with type annotations —i.e., in the Church style— is not available. Geuvers' technique uses a positive reformulation of the counter-example of non-confluence, and the fact that the underlying calculus without typing annotations —i.e., in the Curry style— is confluent.

The underlying Curry style of $\lambda\Pi_{\mathcal{L}}$ is called $\lambda\Pi_{\mathcal{L}}^{\square}$. In this calculus, substitutions do not have type annotations (but abstractions do keep their type annotations). The set of well-formed terms in $\lambda\Pi_{\mathcal{L}}^{\square}$ are the same as in $\lambda\Pi_{\mathcal{L}}$, but substitutions have the following grammar:

$$\textbf{Substitutions}\quad S,T \quad ::= \quad \uparrow^n \mid M\cdot S \mid S\circ T.$$

As in the case of $\lambda\Pi_{\mathcal{L}}$, only meta-variables of terms are enabled in $\lambda\Pi_{\mathcal{L}}^{\square}$. The $\lambda\Pi_{\mathcal{L}}^{\square}$-calculus is obtained by affecting the reduction system $\lambda\Pi_{\mathcal{L}}$ as shown in Fig. 3.1. As expected, we define the $\Pi_{\mathcal{L}}^{\square}$-calculus as $\lambda\Pi_{\mathcal{L}}^{\square}$ without rule (Beta).

The positive reformulation of the confluence counter-example in $\lambda\Pi_{\mathcal{L}}$ states that if two terms are equal without type annotations, then they are convertible via $\equiv_{\lambda\Pi_{\mathcal{L}}}$.

DEFINITION 3.1. *The* erasing mapping $|.| : \lambda\Pi_{\mathcal{L}} \to \lambda\Pi_{\mathcal{L}}^{\square}$ *is defined as follows:*

$$\begin{array}{lcl}
|x| & = & x \quad \text{if } x\in\{\underline{1}, \mathit{Type}, \mathit{Kind}\} \text{ or } x \text{ is a meta-variable}\\
|\Pi_A.B| & = & \Pi_{|A|}.|B|\\
|\lambda_A.B| & = & \lambda_{|A|}.|M|\\
|(M\ N)| & = & (|M|\ |N|)\\
|M[S]| & = & |M|[|S|]\\
|\uparrow^n| & = & \uparrow^n\\
|S\circ T| & = & |S|\circ|T|\\
|M\cdot_A S| & = & |M|\cdot|S|
\end{array}$$

The following are useful properties of the erasing mapping.

LEMMA 3.2 (Erasing properties). *Let x and y be expressions in $\lambda\Pi_{\mathcal{L}}$, w be an expression in $\lambda\Pi_{\mathcal{L}}^{\square}$, R one of the rewrite systems $\lambda\Pi_{\mathcal{L}}$ or $\Pi_{\mathcal{L}}$, and $R^{\square}$ the corresponding rewrite system without type annotations, i.e., $\lambda\Pi_{\mathcal{L}}^{\square}$ or $\Pi_{\mathcal{L}}^{\square}$, then*

1. if $x \xrightarrow{R} y$, then $|x| \xrightarrow{R^{\square}} |y|$ or $|x| = |y|$, *2. if $|x| \xrightarrow{R^{\square}} w$, then there exists w' in $\lambda\Pi_{\mathcal{L}}$ such that $x \xrightarrow{R} w'$ and $|w'| = w$, and*

3. if x is an R-normal form, then $|x|$ is an $R^{\square}$-normal form.

Proof. Properties (1) and (2) are proved by structural induction on x. Property (3) is a consequence of (2). □

LEMMA 3.3 (Positive counter-example). *Let x and y be expressions in $\lambda\Pi_{\mathcal{L}}$, if $|x| = |y|$, then $x \equiv_{\Pi_{\mathcal{L}}} y$, and therefore, $x \equiv_{\lambda\Pi_{\mathcal{L}}} y$.*

Proof. Since $|x| = |y|$, x and y have the same principal constructor. We proceed by structural induction on x. If $x = \lambda_A.M$, $y = \lambda_B.N$, and $|x| = |y|$, then by definition, $\lambda_{|A|}.|M| = \lambda_{|B|}.|N|$ and thus, $|A| = |B|$ and $|M| = |N|$. By induction hypothesis, $A \equiv_{\Pi_{\mathcal{L}}} B$ and $M \equiv_{\Pi_{\mathcal{L}}} N$, and thus, $\lambda_A.M \equiv_{\Pi_{\mathcal{L}}} \lambda_B.N$. In fact, the only interesting case is $x = M \cdot_A S$ and $y = N \cdot_B T$. We get by induction hypothesis:

$$M \equiv_{\Pi_{\mathcal{L}}} N \tag{3.1}$$

$$S \equiv_{\Pi_{\mathcal{L}}} T \tag{3.2}$$

Since the function $|.|$ erases type annotations from substitutions, we do not have by induction hypothesis $A \equiv_{\Pi_{\mathcal{L}}} B$. However, by using the counter-example, we have

$$M \cdot_B S \xleftarrow{\Pi_{\mathcal{L}}^*} (\underline{1} \cdot_A \uparrow^1) \circ (M \cdot_B S) \xrightarrow{\Pi_{\mathcal{L}}^*} M \cdot_A S.$$

We conclude with Eq. 3.1 and Eq. 3.2 that $x = M \cdot_A S \equiv_{\Pi_{\mathcal{L}}} M \cdot_B S \equiv_{\Pi_{\mathcal{L}}} N \cdot_B T = y$. □

A consequence of the reformulation of the counter-example is that, if we erase the type annotations of a term M and then annotate it again with an arbitrary term, we get a term N which is equivalent to M modulo $\equiv_{\lambda\Pi_{\mathcal{L}}}$.

DEFINITION 3.4. *Let A be a term in $\lambda\Pi_{\mathcal{L}}$, the* annotation mapping *$(.)^{\underline{A}} : \lambda\Pi_{\mathcal{L}}^{\square} \to \lambda\Pi_{\mathcal{L}}$ is defined as follows:*

$$\begin{array}{lcl}
x^{\underline{A}} & = & x \quad \text{if } x \in \{\underline{1}, \mathit{Type}, \mathit{Kind}\} \text{ or } x \text{ is a meta-variable} \\
(\Pi_{B_1}.B_2)^{\underline{A}} & = & \Pi_{B_1^{\underline{A}}}.B_2^{\underline{A}} \\
(\lambda_B.M)^{\underline{A}} & = & \lambda_{B^{\underline{A}}}.M^{\underline{A}} \\
(M\ N)^{\underline{A}} & = & (M^{\underline{A}}\ N^{\underline{A}}) \\
(M[S])^{\underline{A}} & = & M^{\underline{A}}[S^{\underline{A}}] \\
(\uparrow^n)^{\underline{A}} & = & \uparrow^n \\
(S \circ T)^{\underline{A}} & = & S^{\underline{A}} \circ T^{\underline{A}} \\
(M \cdot S)^{\underline{A}} & = & M^{\underline{A}} \cdot_A S^{\underline{A}}
\end{array}$$

LEMMA 3.5 (Erasing inverse). *Let x be an expression in $\lambda\Pi_{\mathcal{L}}$ and A be a term in $\lambda\Pi_{\mathcal{L}}$, $x \equiv_{\lambda\Pi_{\mathcal{L}}} |x|^{\underline{A}}$.*

Proof. It is not difficult to show that if w is an expression in $\lambda\Pi_{\mathcal{L}}^{\square}$, then $w = |w^{\underline{A}}|$. Let $w = |x|$, by Lemma 3.3, $x \equiv_{\lambda\Pi_{\mathcal{L}}} |x|^{\underline{A}}$. □

We use the next lemma in the proof of Geuvers' lemma.

LEMMA 3.6. *Let x and y be expressions in $\lambda\Pi_{\mathcal{L}}^{\square}$ and A be a term in $\lambda\Pi_{\mathcal{L}}$, if $x \xrightarrow{\lambda\Pi_{\mathcal{L}}^{\square}} y$, then $x^{\underline{A}} \equiv_{\lambda\Pi_{\mathcal{L}}} y^{\underline{A}}$. Therefore, if $x \xrightarrow{\lambda\Pi_{\mathcal{L}}^{\square *}} y$, then $x^{\underline{A}} \equiv_{\lambda\Pi_{\mathcal{L}}} y^{\underline{A}}$.*

Proof. By induction on the depth of the $\lambda\Pi_{\mathcal{L}}^{\square}$-redex reduced in x. □

The proof of Geuvers' lemma uses a confluence property on the calculus without type annotations. We left the proof of that property (confluence of $\lambda\Pi_{\mathcal{L}}^{\square}$) for the last part of this section.

THEOREM 3.7 (Confluence of $\lambda\Pi_{\mathcal{L}}^{\Box}$). *The $\lambda\Pi_{\mathcal{L}}^{\Box}$-calculus is confluent.*

THEOREM 3.8 (Geuvers' lemma). *Let A_1, B_1, A_2, B_2, M, N be terms in $\lambda\Pi_{\mathcal{L}}$,*

1. *if $\Pi_{A_1}.B_1 \equiv_{\lambda\Pi_{\mathcal{L}}} \Pi_{A_2}.B_2$, then $A_1 \equiv_{\lambda\Pi_{\mathcal{L}}} A_2$ and $B_1 \equiv_{\lambda\Pi_{\mathcal{L}}} B_2$, and*
2. *if $M \equiv_{\lambda\Pi_{\mathcal{L}}} N$, where N is a $\lambda\Pi_{\mathcal{L}}$-normal form, then there exists M' in $\lambda\Pi_{\mathcal{L}}$ such that $M \xrightarrow{\lambda\Pi_{\mathcal{L}}{}^*} M'$ and $|M'| = |N|$.*

Proof. We show only the first case. The second case is similar. By Lemma 3.2(1) and the definition of $|.|$, we have $\Pi_{|A_1|}.|B_1| \equiv_{\lambda\Pi_{\mathcal{L}}^{\Box}} \Pi_{|A_2|}.|B_2|$. Since $\lambda\Pi_{\mathcal{L}}^{\Box}$ is confluent (Theorem 3.7), there exists M in $\lambda\Pi_{\mathcal{L}}^{\Box}$ such that $\Pi_{|A_1|}.|B_1| \xrightarrow{\lambda\Pi_{\mathcal{L}}^{\Box *}} M$ and $\Pi_{|A_2|}.|B_2| \xrightarrow{\lambda\Pi_{\mathcal{L}}^{\Box *}} M$. But there is no $\lambda\Pi_{\mathcal{L}}^{\Box}$-redex with a product as the main constructor, so M has the form $\Pi_A.B$ where $|A_1| \xrightarrow{\lambda\Pi_{\mathcal{L}}^{\Box *}} A$, $|B_1| \xrightarrow{\lambda\Pi_{\mathcal{L}}^{\Box *}} B$, $|A_2| \xrightarrow{\lambda\Pi_{\mathcal{L}}^{\Box *}} A$, and $|B_2| \xrightarrow{\lambda\Pi_{\mathcal{L}}^{\Box *}} B$. By Lemma 3.5 and Lemma 3.6, for any $\lambda\Pi_{\mathcal{L}}$-term N, $A_1 \equiv_{\lambda\Pi_{\mathcal{L}}} |A_1|^{\underline{N}} \equiv_{\lambda\Pi_{\mathcal{L}}} A^{\underline{N}}$, $B_1 \equiv_{\lambda\Pi_{\mathcal{L}}} |B_1|^{\underline{N}} \equiv_{\lambda\Pi_{\mathcal{L}}} B^{\underline{N}}$, $A_2 \equiv_{\lambda\Pi_{\mathcal{L}}} |A_2|^{\underline{N}} \equiv_{\lambda\Pi_{\mathcal{L}}} A^{\underline{N}}$, and $B_2 \equiv_{\lambda\Pi_{\mathcal{L}}} |B_2|^{\underline{N}} \equiv_{\lambda\Pi_{\mathcal{L}}} B^{\underline{N}}$. Therefore, $A_1 \equiv_{\lambda\Pi_{\mathcal{L}}} A_2$ and $B_1 \equiv_{\lambda\Pi_{\mathcal{L}}} B_2$. □

The rest of this section addresses the proof of confluence of the $\lambda\Pi_{\mathcal{L}}^{\Box}$-calculus (Theorem 3.7).

First, we prove that the $\Pi_{\mathcal{L}}^{\Box}$-calculus —$\lambda\Pi_{\mathcal{L}}^{\Box}$ without (Beta)— is terminating and confluent.

LEMMA 3.9 (Termination of $\Pi_{\mathcal{L}}^{\Box}$). *$\Pi_{\mathcal{L}}^{\Box}$ is a terminating rewrite system.*

Proof. Since any reduction in $\Pi_{\mathcal{L}}^{\Box}$ can be properly simulated in $\Pi_{\mathcal{L}}$ (Lemma 3.2(2)), any infinite reduction in $\Pi_{\mathcal{L}}^{\Box}$ corresponds to some infinite reduction in $\Pi_{\mathcal{L}}$. But $\Pi_{\mathcal{L}}$ is terminating (Lemma 2.1), thus $\Pi_{\mathcal{L}}^{\Box}$ is terminating. □

LEMMA 3.10 (Confluence of $\Pi_{\mathcal{L}}^{\Box}$). *The $\Pi_{\mathcal{L}}^{\Box}$-calculus is confluent.*

Proof. We mechanically check, e.g., by using the RRL system [23], that the $\Pi_{\mathcal{L}}^{\Box}$-rewrite system has the following critical pairs:

- (Id)-(Clos)

$$M[S] \xleftarrow{\Pi_{\mathcal{L}}^{\Box +}} M[S][\uparrow^0] \xrightarrow{\Pi_{\mathcal{L}}^{\Box +}} M[S \circ \uparrow^0]$$

- (Clos)-(Clos)

$$M[(S_1 \circ S_2) \circ T] \xleftarrow{\Pi_{\mathcal{L}}^{\Box +}} M[S_1][S_2][T] \xrightarrow{\Pi_{\mathcal{L}}^{\Box +}} M[S_1 \circ (S_2 \circ T)]$$

- (Shift0)-(Map)

$$S \xleftarrow{\Pi_{\mathcal{L}}^{\Box +}} (\underline{1} \cdot \uparrow^1) \circ S \xrightarrow{\Pi_{\mathcal{L}}^{\Box}} \underline{1}[S] \cdot (\uparrow^1 \circ S)$$

- (ShiftS)-(Map)

$$\uparrow^n \circ S \xleftarrow{\Pi_{\mathcal{L}}^{\Box}} (\underline{1}[\uparrow^n] \cdot \uparrow^{n+1}) \circ S \xrightarrow{\Pi_{\mathcal{L}}^{\Box +}} \underline{1}[\uparrow^n \circ S] \cdot (\uparrow^{n+1} \circ S)$$

- (Lambda)-(Clos) and (Pi)-(Clos)
  Let $S_1 = \underline{1} \cdot ((S \circ \uparrow^1) \circ (\underline{1} \cdot (T \circ \uparrow^1)))$ and $S_2 = \underline{1} \cdot ((S \circ T) \circ \uparrow^1)$,

$$\lambda_{A[S \circ T]}.M[S_1] \xleftarrow{\Pi_{\mathcal{L}}^{\Box +}} (\lambda_A.M)[S][T] \xrightarrow{\Pi_{\mathcal{L}}^{\Box +}} \lambda_{A[S \circ T]}.M[S_2]$$

$$\Pi_{A[S \circ T]}.B[S_1] \xleftarrow{\Pi_{\mathcal{L}}^{\Box +}} (\Pi_A.B)[S][T] \xrightarrow{\Pi_{\mathcal{L}}^{\Box +}} \Pi_{A[S \circ T]}.B[S_2]$$

These critical pairs are $\Pi_{\mathcal{L}}^{\Box}$-joinable (we recall that only meta-variables of terms are admitted). Using an extension to the Critical Pair lemma proposed in [33] (based on similar extensions originally presented in

$$\frac{}{x \longrightarrow x}\ (\mathrm{Refl}_{\|}) \qquad \frac{A \longrightarrow B \quad M \longrightarrow N}{\lambda_A.M \longrightarrow \lambda_B.N}\ (\mathrm{Lambda}_{\|})$$

$$\frac{A_1 \longrightarrow B_1 \quad A_2 \longrightarrow B_2}{\Pi_{A_1}.B_1 \longrightarrow \Pi_{A_2}.B_2}\ (\mathrm{Pi}_{\|}) \qquad \frac{M \longrightarrow N \quad S \longrightarrow T}{M[S] \longrightarrow N[T]}\ (\mathrm{Clos}_{\|})$$

$$\frac{M_1 \longrightarrow M_2 \quad N_1 \longrightarrow N_2}{(M_1\ N_1) \longrightarrow (M_2\ N_2)}\ (\mathrm{Application}_{\|}) \qquad \frac{M \longrightarrow N \quad S \longrightarrow T}{M \cdot S \longrightarrow N \cdot T}\ (\mathrm{Cons}_{\|})$$

$$\frac{S_1 \longrightarrow S_2 \quad T_1 \longrightarrow T_2}{S_1 \circ T_1 \longrightarrow S_2 \circ T_2}\ (\mathrm{Comp}_{\|}) \qquad \frac{M_1 \longrightarrow M_2 \quad N_1 \longrightarrow N_2}{(\lambda_A.M_1\ N_1) \longrightarrow M_2[N_2 \cdot \uparrow^0]}\ (\mathrm{Beta}_{\|})$$

FIG. 3.2. *The parallelization of (Beta)*

[22, 40]), we conclude that $\Pi_{\mathcal{L}}^{\Box}$ is locally confluent. Therefore, by Newman's lemma and Lemma 3.9, $\Pi_{\mathcal{L}}^{\Box}$ is confluent. □

The confluence proof of the $\lambda\Pi_{\mathcal{L}}^{\Box}$-calculus uses a general method proposed in [45] to prove confluence of abstract relations: the Yokouchi-Hikita's lemma. This method shows to be suitable for left-linear calculi of explicit substitutions [7, 37, 33].

LEMMA 3.11 (Yokouchi-Hikita's lemma). *Let R and S be two relations defined on a set X such that: 1) R is confluent and terminating, 2) S is strongly confluent, and 3) S and R commute in the following way: for any $x, y, z \in X$, if $x \xrightarrow{R} y$ and $x \xrightarrow{S} z$, then there exists $w \in X$ such that $y \xrightarrow{R^*SR^*} w$ and $z \xrightarrow{R^*} w$. Then the relation R^*SR^* is confluent.*

Proof. See [7]. □

We take the set of $\lambda\Pi_{\mathcal{L}}^{\Box}$-expressions as X, $\Pi_{\mathcal{L}}^{\Box}$ as R and $B_{\|}$ as S, where $B_{\|}$ is the parallelization of (Beta) defined in Fig. 3.2.

LEMMA 3.12. *$\Pi_{\mathcal{L}}^{\Box}$ commutes over $B_{\|}$, i.e., if x reduces in one $\Pi_{\mathcal{L}}^{\Box}$-step to y, and in one $B_{\|}$-step to z, then there exists w such that $y \xrightarrow{\Pi_{\mathcal{L}}^{\Box *} B_{\|} \Pi_{\mathcal{L}}^{\Box *}} w$ and $z \xrightarrow{\Pi_{\mathcal{L}}^{\Box *}} w$.*

Proof. By case analysis on the redex reduced in x. □

We are now ready to prove the confluence property of $\lambda\Pi_{\mathcal{L}}^{\Box}$.

Theorem 3.7. The $\lambda\Pi_{\mathcal{L}}^{\Box}$-calculus is confluent.

Proof. We verify that $\Pi_{\mathcal{L}}^{\Box}$ and $B_{\|}$ satisfy the conditions of Yokouchi-Hikita's lemma, that is,

1. $\Pi_{\mathcal{L}}^{\Box}$ is terminating and confluent (Lemma 3.9 and Lemma 3.10),
2. $B_{\|}$ is strongly confluent, since (Beta) by itself is a left linear system with no critical pairs (c.f. [19]), and
3. $\Pi_{\mathcal{L}}^{\Box}$ commutes over $B_{\|}$ (Lemma 3.12).

Therefore, $\Pi_{\mathcal{L}}^{\Box *} B_{\|} \Pi_{\mathcal{L}}^{\Box *}$ is confluent.

Note that $\lambda\Pi_{\mathcal{L}}^{\Box} \subseteq \Pi_{\mathcal{L}}^{\Box *} B_{\|} \Pi_{\mathcal{L}}^{\Box *} \subseteq \lambda\Pi_{\mathcal{L}}^{\Box *}$. Let x be an expression in $\lambda\Pi_{\mathcal{L}}^{\Box}$. If $x \xrightarrow{\lambda\Pi_{\mathcal{L}}^{\Box *}} y$ and $x \xrightarrow{\lambda\Pi_{\mathcal{L}}^{\Box *}} z$, then there exists w such that $y \xrightarrow{(\Pi_{\mathcal{L}}^{\Box *} B_{\|} \Pi_{\mathcal{L}}^{\Box *})^*} w$ and $z \xrightarrow{(\Pi_{\mathcal{L}}^{\Box *} B_{\|} \Pi_{\mathcal{L}}^{\Box *})^*} w$. So, $y \xrightarrow{\lambda\Pi_{\mathcal{L}}^{\Box *}} w$ and $z \xrightarrow{\lambda\Pi_{\mathcal{L}}^{\Box *}} w$. □

4. Elementary Typing Properties.

The elementary typing properties of $\lambda\Pi_{\mathcal{L}}$ are

- *Sort soundness*: the type of a term is a valid sort.
- *Type uniqueness*: the type of a term is unique module $\equiv_{\lambda\Pi_{\mathcal{L}}}$.
- *Subject reduction*: the $\lambda\Pi_{\mathcal{L}}$-rewrite system preserves typing.

- *Soundness*: there always exists a path of well-typed terms between equivalent well-typed terms.

We use Geuvers' lemma to prove the last two of the above properties.

THEOREM 4.1 (Sort soundness).

1. *If* $\Sigma; \Gamma \vdash M : A$, *then* $A = Kind$ *or* $\Sigma; \Gamma \vdash A : s$, $s \in \{Kind, Type\}$, *and*
2. *if* $\Sigma; \Gamma \vdash S \rhd \Delta$ *then* $\Sigma; \Delta$.

Proof. By induction on the typing derivation. □

THEOREM 4.2 (Type uniqueness). *Let* Γ_1 *and* Γ_2 *be such that* $\Gamma_1 \equiv_{\lambda\Pi_{\mathcal{L}}} \Gamma_2$,

1. *if* $\Sigma; \Gamma_1 \vdash M : A$ *and* $\Sigma; \Gamma_2 \vdash M : B$, *then* $A \equiv_{\lambda\Pi_{\mathcal{L}}} B$, *and*
2. *if* $\Sigma; \Gamma_1 \vdash S \rhd \Delta_1$ *and* $\Sigma; \Gamma_2 \vdash S \rhd \Delta_2$, *then* $\Delta_1 \equiv_{\lambda\Pi_{\mathcal{L}}} \Delta_2$.

Proof. By simultaneous structural induction on M and S. □

THEOREM 4.3 (Subject reduction). *The* $\lambda\Pi_{\mathcal{L}}$*-calculus preserves typing, if* $x \xrightarrow{\lambda\Pi_{\mathcal{L}}^*} y$, *for an expression* x, *then*

1. *if* x *is a term and* $\Sigma; \Gamma \vdash x : A$, *then* $\Sigma; \Gamma \vdash y : A$, *and*
2. *if* x *is a substitution and* $\Sigma; \Gamma \vdash x \rhd \Delta$, *then* $\Sigma; \Gamma \vdash y \rhd \Delta$.

Proof. We show that typing is preserved for one-step reductions (i.e., $\xrightarrow{\lambda\Pi_{\mathcal{L}}}$), and therefore, it is also for the reflexive and transitive closure (i.e., $\xrightarrow{\lambda\Pi_{\mathcal{L}}^*}$). Let $x \xrightarrow{\lambda\Pi_{\mathcal{L}}} y$ be a one-step reduction. We proceed by induction on the depth of the redex reduced in x.

In the initial case, x is reduced at the top level, and we proceed by case analysis. We show the case of rule (Beta):

Let $\Sigma; \Gamma \vdash (\lambda_A.M\ N) : B$. We show $\Sigma; \Gamma \vdash M[N \cdot_A \uparrow^0] : B$.

We have:

1. (a) $\Sigma; \Gamma \vdash \lambda_A.M : \Pi_{A_1}.B_1$, (b) $\Sigma; \Gamma \vdash N : A_1$, and (c) $B \equiv_{\lambda\Pi_{\mathcal{L}}} B_1[N \cdot_{A_1} \uparrow^0]$, by inversion of rule (Appl) applied to the hypothesis.
2. (a) $\Sigma; \Gamma \vdash A : Type$, (b) $\Sigma; A.\Gamma \vdash M : B_2$, (c) $\Sigma; A.\Gamma \vdash B_2 : s_2$, $s_2 \in \{Kind, Type\}$, and (d) $\Pi_A.B_2 \equiv_{\lambda\Pi_{\mathcal{L}}} \Pi_{A_1}.B_1$, by inversion of rule (Abs) applied to (1-a).
3. (a) $A \equiv_{\lambda\Pi_{\mathcal{L}}} A_1$ and (b) $B_2 \equiv_{\lambda\Pi_{\mathcal{L}}} B_1$, by Geuvers' lemma (Theorem 3.8) applied to (2-d).
4. $\Sigma; \Gamma \vdash N : A$, by rule (Conv) applied to (1-b), (2-a), and (3-a).
5. $\Sigma; \Gamma \vdash N \cdot_A \uparrow^0 \rhd A.\Gamma$, by rule (Cons) applied to (4), (2-a), and $\Sigma; \Gamma \vdash \uparrow^0 \rhd \Gamma$.
6. $B_2[N \cdot_A \uparrow^0] \equiv_{\lambda\Pi_{\mathcal{L}}} B_1[N \cdot_A \uparrow^0] \equiv_{\lambda\Pi_{\mathcal{L}}} B_1[N \cdot_{A_1} \uparrow^0] \equiv_{\lambda\Pi_{\mathcal{L}}} B$, by (1-c) and (3).
7. $\Sigma; \Gamma \vdash B : s_1$, $s_1 \in \{Kind, Type\}$, by sort soundness (Theorem 4.1) applied to the hypothesis. Note that the case $s = Kind$ is not possible.

Therefore, we have the derivation

$$\dfrac{\dfrac{\begin{array}{ll}\Sigma; A.\Gamma \vdash M : B_2 & (2\text{-b})\\ \Sigma; A.\Gamma \vdash B_2 : s_2 & (2\text{-c})\\ \Sigma; \Gamma \vdash N \cdot_A \uparrow^0 \rhd A.\Gamma & (5)\end{array}}{\Sigma; \Gamma \vdash M[N \cdot_A \uparrow^0] : B_2[N \cdot_A \uparrow^0]}\,(\text{Clos}) \qquad (6) \quad (7)}{\Sigma; \Gamma \vdash M[N \cdot_A \uparrow^0] : B}\,(\text{Conv})$$

The other cases are similar. The induction step cases do not present any difficulty. □

Sometimes the conversion rule (Conv) is expressed as [14]:

$$\frac{\begin{array}{c}\Gamma \vdash M : A \\ \Gamma \vdash B : s \\ s \in \{\mathit{Kind}, \mathit{Type}\} \\ A \longrightarrow B \text{ or } B \longrightarrow A\end{array}}{\Gamma \vdash M : B}\ (\text{Conv'})$$

Rule (Conv) seems to be more general than rule (Conv'). In fact, the latter one allows conversions of types only via a path of well-typed terms. Geuvers and Werner [14] define a *type system to be sound* if the convertibility of terms remains in the set of well-typed terms. In sound systems, rules (Conv) and (Conv') are equivalent.

We use the following lemma in the soundness proof of the $\lambda\Pi_{\mathcal{L}}$-system.

LEMMA 4.4. *Let x, y be $\lambda\Pi_{\mathcal{L}}$-expressions in $\Pi_{\mathcal{L}}$-normal form such that $|x| = |y|$, if x and y are well-typed expressions, then they are convertible via a path of well-typed expressions.*

Proof. By structural induction on x and y. □

THEOREM 4.5 (Soundness). *If $\Sigma;\Gamma \vdash M : A$, $\Sigma;\Gamma \vdash N : B$ and $M \equiv_{\lambda\Pi_{\mathcal{L}}} N$, then M and N are convertible via a path of well-typed terms.*

Proof. From Lemma 3.2(1), we have $|M| \equiv_{\lambda\Pi_{\mathcal{L}}} |N|$. The confluence property of $\lambda\Pi_{\mathcal{L}}^{\Box}$ states that there exists $x \in \lambda\Pi_{\mathcal{L}}^{\Box}$ such that $|M| \xrightarrow{\lambda\Pi_{\mathcal{L}}^{\Box}{}^*} x$ and $|N| \xrightarrow{\lambda\Pi_{\mathcal{L}}^{\Box}{}^*} x$. By Lemma 3.2(2), there exist M_1, N_1 in $\lambda\Pi_{\mathcal{L}}$ such that $M \xrightarrow{\lambda\Pi_{\mathcal{L}}{}^*} M_1$, $N \xrightarrow{\lambda\Pi_{\mathcal{L}}{}^*} N_1$, and $|M_1| = |N_1| = x$. Since $\Pi_{\mathcal{L}}$ is terminating (Lemma 2.1), there exist M_2, N_2 $\Pi_{\mathcal{L}}$-normal forms such that $M_1 \xrightarrow{\Pi_{\mathcal{L}}{}^*} M_2$, $N_1 \xrightarrow{\Pi_{\mathcal{L}}{}^*} N_2$. By the subject reduction property (Theorem 4.3), $\Sigma;\Gamma \vdash M_2 : A$ and $\Sigma;\Gamma \vdash N_2 : B$, and all the terms in both reductions are well-typed.

Now, from Lemma 3.2(1), we have $x \xrightarrow{\Pi_{\mathcal{L}}^{\Box}{}^*} |M_2|$ and $x \xrightarrow{\Pi_{\mathcal{L}}^{\Box}{}^*} |N_2|$. But M_2 and N_2 are $\Pi_{\mathcal{L}}$-normal forms, thus, by Lemma 3.2(3), $|M_2|$ and $|N_2'|$ are $\Pi_{\mathcal{L}}^{\Box}$-normal forms. Since $\Pi_{\mathcal{L}}^{\Box}$ is confluent, $|M_2| = |N_2|$. By Lemma 4.4, M_2 and N_2 are convertible via a path of well-typed terms. Therefore, M and N are convertible via a path of well-typed terms. □

A direct consequence of typing soundness and subject reduction is the following property.

LEMMA 4.6. *If $\Sigma;\Gamma \vdash M_1 : A_1$, $\Sigma;\Gamma \vdash M_2 : A_2$, and $M_1 \equiv_{\lambda\Pi_{\mathcal{L}}} M_2$, then $A_1 \equiv_{\lambda\Pi_{\mathcal{L}}} A_2$.*

Proof. By induction on the length of the paths of well-typed expressions converting M_1 to M_2. □

5. The Main Properties: Weak Normalization and Confluence.

In this section we address the proof of the main properties of $\lambda\Pi_{\mathcal{L}}$ on well-typed expressions: weak normalization and confluence.

5.1. Weak normalization.

The $\lambda\Pi_{\mathcal{L}}$-calculus does not preserve strong normalization of $\lambda\Pi$. In fact, the counterexample shown in [30] for $\lambda\sigma$ may be reproduced in $\lambda\Pi_{\mathcal{L}}$ with some minor modifications.

Nevertheless, we prove that $\lambda\Pi_{\mathcal{L}}$ is weakly normalizing on well-typed expressions, i.e., there exists a strategy to find $\lambda\Pi_{\mathcal{L}}$-normal forms on well-typed expressions. In particular, we propose a proof of strong normalization of the strategy that performs one step of (Beta) followed by a $\Pi_{\mathcal{L}}$-normalization.

We use the standard technique of reducibility, originally due to Tait for the simply-typed λ-calculus [42], and then extended by Girard to the system F (the λ-calculus of second-order) [15]. From the diverse proofs of termination using a reducibility notion, we follow the presentation given in [12] for the Calculus of Constructions, which is based on saturated sets. We adapt this proof for the $\lambda\Pi_{\mathcal{L}}$-calculus. In order to avoid some technical problems due to the non-confluence of the calculus with type annotations (not necessarily well-typed), we define saturated sets in a slightly different way. However, the structure of the proofs is the same.

We use $(x){\downarrow}_{\Pi_{\mathcal{L}}}$ as a shorthand for the set of $\Pi_{\mathcal{L}}$-normal forms of x. The set containing all the $\Pi_{\mathcal{L}}$-normal forms of $\lambda\Pi_{\mathcal{L}}$ is denoted by $\mathcal{NF}$.

Definition 5.1. *Let $x, y \in \mathcal{NF}$, we say that x* $\beta\Pi_{\mathcal{L}}$-reduces *to y, denoted by $x \xrightarrow{\beta\Pi_{\mathcal{L}}} y$, if $x \xrightarrow{(Beta)} w$ and $y \in (w){\downarrow}_{\Pi_{\mathcal{L}}}$.* Notice that the set of $\beta\Pi_{\mathcal{L}}$-normal forms is equal to the set of $\lambda\Pi_{\mathcal{L}}$-normal forms, and that $x \xrightarrow{\beta\Pi_{\mathcal{L}}} y$ implies $x \xrightarrow{\lambda\Pi_{\mathcal{L}}{}^*} y$. In fact, we will show that $\beta\Pi_{\mathcal{L}}$ is strongly normalizing on well-typed expressions, and therefore, $\lambda\Pi_{\mathcal{L}}$ is weakly normalizing on well-typed expressions.

We denote by $\mathcal{SN}$ the set of $\beta\Pi_{\mathcal{L}}$-strongly normalizing expressions of $\mathcal{NF}$.

Definition 5.2. *Let M be a term in $\mathcal{NF}$. The term M is* neutral *if it does not have the form $\lambda_A.N$. The set of neutral terms is denoted by $\mathcal{NT}$.*

Definition 5.3. *Let x be in $\mathcal{NF}$. The set of* annotations *of x, denoted by $\aleph(x)$, is defined inductively as follows:*

$$\begin{array}{lcl}
\aleph(x) & = & \emptyset \quad \textit{if } x \in \{\textit{Kind}, \textit{Type}, \underline{1}\} \textit{ or } x = \uparrow^n \textit{ or } x \textit{ is a meta-variable} \\
\aleph(\Pi_A.B) & = & \aleph(A) \cup \aleph(B) \\
\aleph(\lambda_A.M) & = & \aleph(A) \cup \aleph(M) \\
\aleph(M\ N) & = & \aleph(M) \cup \aleph(N) \\
\aleph(M[S]) & = & \aleph(M) \cup \aleph(S) \\
\aleph(S \circ T) & = & \aleph(S) \cup \aleph(T) \\
\aleph(M \cdot_A S) & = & \{A\} \cup \aleph(M) \cup \aleph(S)
\end{array}$$

Definition 5.4. *A set of terms $\Lambda \subseteq \mathcal{NF}$ is* saturated *if*

1. *$\Lambda \subseteq \mathcal{SN}$,*
2. *if $M \in \Lambda$ and $M \xrightarrow{\beta\Pi_{\mathcal{L}}} N$, then $N \in \Lambda$,*
3. *if $M \in \mathcal{NT}$, and whenever the reduction of a $\beta\Pi_{\mathcal{L}}$-redex of M leads to a term $N \in \Lambda$, then $M \in \Lambda$, and*
4. *if $M \in \Lambda$, $|M| = |N|$, and $\aleph(N) \subseteq \mathcal{SN}$, then $N \in \Lambda$.*

The set of saturated sets is denoted by **SAT**.

The following corollary is a trivial consequence of Def. 5.4(3).

Corollary 5.5. *Let $M \in \mathcal{NT}$ such that M is a $\beta\Pi_{\mathcal{L}}$-normal form, for any $\Lambda \in$ **SAT**, $M \in \Lambda$.*

The following lemmas show particular cases of terms that are in saturated sets.

Lemma 5.6. *For any $\Lambda \in$ **SAT**, substitution $S \in \mathcal{SN}$, and meta-variable X, we have $(X[S]){\downarrow}_{\Pi_{\mathcal{L}}} \subseteq \Lambda$.*

Proof. Let $\Lambda \in$ **SAT** and $M \in (X[S]){\downarrow}_{\Pi_{\mathcal{L}}}$. Since M is neutral it suffices to consider the reductions of M (Def. 5.4(3)). We reason by induction on $\nu(S)$[3]. Only two reductions are possible:

- $M \xrightarrow{\beta\Pi_{\mathcal{L}}} X$, and by Corollary 5.5, $X \in \Lambda$.
- $M \xrightarrow{\beta\Pi_{\mathcal{L}}} X[T]$ where $S \xrightarrow{\beta\Pi_{\mathcal{L}}} T$. By hypothesis, $T \in \mathcal{SN}$, and $\nu(S) > \nu(T)$, so by induction hypothesis, $(X[T]){\downarrow}_{\Pi_{\mathcal{L}}} \subseteq \Lambda$.

In both cases, M reduces to terms in Λ, thus, $M \in \Lambda$. □

Lemma 5.7. *For any $\Lambda \in$ **SAT**, and terms $A, B \in \mathcal{SN}$, $\Pi_A.B \in \Lambda$.*

Proof. The term $\Pi_A.B$ is neutral. By Def. 5.4(3) it suffices to consider the reductions of $\Pi_A.B$. We reason by induction on $\nu(A) + \nu(B)$. □

Lemma 5.8. $\mathcal{SN} \in$ **SAT**.

Proof. We verify the following conditions (Def. 5.4).

[3] "If x is strongly normalizing, $\nu(x)$ is a number which bounds the length of every normalization sequence beginning with x" [16].

1. $\mathcal{SN} \subseteq \mathcal{SN}$.
2. If $M \in \mathcal{SN}$ and $M \xrightarrow{\beta\Pi_{\mathcal{L}}} N$, then $N \in \mathcal{SN}$.
3. If $M \in \mathcal{NT}$, and whenever the reduction of a $\beta\Pi_{\mathcal{L}}$-redex of M leads to a term $N \in \mathcal{SN}$, then $M \in \mathcal{SN}$.
4. If $M \in \mathcal{SN}$, $|M| = |N|$, and $\aleph(N) \subseteq \mathcal{SN}$, then $N \in \mathcal{SN}$.

□

DEFINITION 5.9. *If* $\Lambda, \Lambda' \in \mathbf{SAT}$, *we define the set*

$$\Lambda \to \Lambda' = \{M \in \mathcal{NF} \mid \forall N \in \Lambda, (M\ N) \in \Lambda'\}.$$

LEMMA 5.10. **SAT** *is closed under function spaces, i.e., if* $\Lambda, \Lambda' \in \mathbf{SAT}$, *then* $\Lambda \to \Lambda' \in \mathbf{SAT}$.

Proof. We verify the conditions in Def. 5.4:

1. $\Lambda \to \Lambda' \subseteq \mathcal{SN}$:
   Let M be in $\Lambda \to \Lambda'$. By Def. 5.9 and Def. 5.4(1), $(M\ N) \in \Lambda' \subseteq \mathcal{SN}$ for all $N \in \Lambda$. Thus, $M \in \mathcal{SN}$.
2. If $M \in \Lambda \to \Lambda'$ and $M \xrightarrow{\beta\Pi_{\mathcal{L}}} N$, then $N \in \Lambda \to \Lambda'$.
   Let N_1 be in Λ. We show that $(N\ N_1) \in \Lambda'$. By hypothesis, $(M\ N_1) \in \Lambda'$ and $(M\ N_1) \xrightarrow{\beta\Pi_{\mathcal{L}}} (N\ N_1)$. Thus, $(N\ N_1) \in \Lambda'$ by Def. 5.4(2).
3. If $M \in \mathcal{NT}$, and whenever the reduction of a $\beta\Pi_{\mathcal{L}}$-redex of M leads to a term $N \in \Lambda \to \Lambda'$, then $M \in \Lambda \to \Lambda'$.
   Let N_1 be in Λ, we show that $(M\ N_1) \in \Lambda'$. Since $(M\ N_1) \in \mathcal{NT}$, it suffices by Def. 5.4(3) to prove that if $(M\ N_1) \xrightarrow{\beta\Pi_{\mathcal{L}}} N_2$, then $N_2 \in \Lambda'$. We have $N_1 \in \Lambda \subseteq \mathcal{SN}$. We reason by induction on $\nu(N_1)$. Since $M \in \mathcal{NT}$, $(M\ N_1)$ $\beta\Pi_{\mathcal{L}}$-reduces in one step to
   - $(M_1\ N_1)$, with $M \xrightarrow{\beta\Pi_{\mathcal{L}}} M_1$. By hypotheses, $M_1 \in \Lambda \to \Lambda'$ and $N_1 \in \Lambda$, thus $(M_1\ N_1) \in \Lambda'$.
   - $(M\ N_2)$, with $N_1 \xrightarrow{\beta\Pi_{\mathcal{L}}} N_2$. By Def. 5.4(2), $N_2 \in \Lambda$ and $\nu(N_2) < \nu(N_1)$, thus, by induction hypothesis, $(M\ N_2) \in \Lambda'$.

   In both cases, $(M\ N_1)$ reduces to terms in Λ'. Hence, $(M\ N_1) \in \Lambda'$
4. If $M \in \Lambda \to \Lambda'$, $|M| = |N|$, and $\aleph(N) \subseteq \mathcal{SN}$, then $N \in \Lambda \to \Lambda'$.
   Let N_1 be in Λ. We show that $(N\ N_1) \in \Lambda'$. By hypothesis, $(M\ N_1) \in \Lambda'$, but also, $|(M\ N_1)| = |(N\ N_1)|$. By Def. 5.4(4), it suffices to show that $\aleph(N\ N_1) \subseteq \mathcal{SN}$. Since $N_1 \in \Lambda \subseteq \mathcal{SN}$, we have $\aleph(N_1) \subseteq \mathcal{SN}$. Therefore, $\aleph(N\ N_1) = \aleph(N) \cup \aleph(N_1) \subseteq \mathcal{SN}$.

□

The next step in the proof is the interpretation of types.

DEFINITION 5.11. *The* type interpretation function *of terms in* $\lambda\Pi_{\mathcal{L}}$ *is defined inductively as follows:*

$$\begin{array}{lcll} [\![x]\!] & = & \mathcal{SN} & \textit{if } x \in \{\textit{Kind}, \textit{Type}, \underline{1}\} \textit{ or } x \textit{ is a meta-variable} \\ [\![M[S]]\!] & = & [\![M]\!] & \\ [\![(M\ N)]\!] & = & [\![M]\!] & \\ [\![\lambda_A.B]\!] & = & [\![B]\!] & \\ [\![\Pi_A.B]\!] & = & [\![A]\!] \to [\![B]\!] & \end{array}$$

We have the following corollary of Lemma 5.10.

COROLLARY 5.12. *For any term* M, $[\![M]\!] \in \mathbf{SAT}$.

Lists of types, i.e., contexts, are interpreted by a set of explicit substitutions.

Definition 5.13. *The* valuations *of* Γ, *denoted by* $[\![\Gamma]\!]$, *is a set of substitutions in* $\mathcal{NF}$ *defined inductively on* Γ *as follows:*

$$\begin{array}{lcl} [\![\epsilon]\!] & = & \{\uparrow^n \mid \textit{for any natural } n\} \\ [\![A.\Delta]\!] & = & [\![\epsilon]\!] \cup \{M \cdot_B S \in \mathcal{NF} \mid M \in [\![B]\!], S \in [\![\Delta]\!], B \in \mathcal{SN}, [\![A]\!] = [\![B]\!]\} \end{array}$$

Lemma 5.14. *For any* Γ, $[\![\Gamma]\!] \subseteq \mathcal{SN}$.

Proof. We show by structural induction on S that if $S \in [\![\Gamma]\!]$, then $S \in \mathcal{SN}$. □

Definition 5.15. *Let M be a term in $\mathcal{NF}$ and S be a substitution in $\mathcal{NF}$. We define*

1. Γ satisfies *that M is of type A, denoted by* $\Gamma \models M : A$, *if and only if* $(M[T])\downarrow_{\Pi_\mathcal{L}} \subseteq [\![A]\!]$ *for any* $T \in [\![\Gamma]\!]$.
2. Γ satisfies *that S is of type Δ, denoted by* $\Gamma \models S \triangleright \Delta$, *if and only if* $(S \circ T)\downarrow_{\Pi_\mathcal{L}} \subseteq [\![\Delta]\!]$ *for any* $T \in [\![\Gamma]\!]$.

We are almost ready to prove the key property which leads to the strong normalization property of $\beta\Pi_\mathcal{L}$. It states that if $\Gamma \models M : A$, then $\Gamma \vdash M : A$. Before that, we need some more technical lemmas.

Lemma 5.16. *Let A be a term in $\mathcal{SN}$. For all substitutions $S \in [\![\Gamma]\!]$ and term $M \in [\![A]\!]$,* $(M \cdot_A S)\downarrow_{\Pi_\mathcal{L}} \subseteq [\![A.\Gamma]\!]$.

Proof. Note that $M \cdot_A S$ is not necessarily in $\mathcal{NF}$. But there are two cases: $(M \cdot_A S)\downarrow_{\Pi_\mathcal{L}} = \{M \cdot_A S\}$ or $(M \cdot_A S)\downarrow_{\Pi_\mathcal{L}} = \{\uparrow^n\}$. In both cases we verify that $(M \cdot_A S)\downarrow_{\Pi_\mathcal{L}} \subseteq [\![A.\Gamma]\!]$. □

Lemma 5.17. *Let M a term in $\mathcal{NF}$, if $\Sigma;\Gamma \vdash M : A$ and $\Sigma;\Gamma \vdash A : Type$, then* $[\![M]\!] = \mathcal{SN}$.

Proof. By structural induction on M. We show the case where $M = (M_1\ M_2)$, the other cases are similar. We have:

1. (a) $\Sigma;\Gamma \vdash M_1 : \Pi_{A_1}.B_1$, (b) $\Sigma;\Gamma \vdash (M_1\ M_2) : B_1[M_2 \cdot_{A_1} \uparrow^0]$, and (c) $A \equiv_{\lambda\Pi_\mathcal{L}} B_1[M_2 \cdot_{A_1} \uparrow^0]$, by inversion of rule (Appl) applied to the hypothesis.
2. (a) $\Sigma;\Gamma \vdash A_1 : Type$ and (b) $\Sigma; A_1.\Gamma \vdash B_1 : s_1$, $s_1 \in \{Kind, Type\}$, by inversion of rule (Prod) applied to (1-a).
3. $\Sigma;\Gamma \vdash B_1[M_2 \cdot_{A_1} \uparrow^0] : s_2$, $s_2 \in \{Kind, Type\}$, by sort soundness (Theorem 4.1) applied to (1-b).
4. $s_2 \equiv_{\lambda\Pi_\mathcal{L}} Type$, by Lemma 4.6 applied to $\Sigma;\Gamma \vdash A : Type$, (1-c), and (3).
5. $s_2 = Type$, by Geuvers' lemma (Theorem 3.8) applied to (4).
6. $s_1 = Type$, by (2-b), (3), and (5).

Then, applying rule (Prod) to (2) and (6), we get $\Sigma;\Gamma \vdash \Pi_{A_1}.B_1 : Type$. By Def. 5.11 and induction hypothesis, $[\![(M_1\ M_2)]\!] = [\![M_1]\!] = \mathcal{SN}$. □

Lemma 5.18. *Let M be a term in $\mathcal{NF}$ and S a substitution in $\mathcal{NT}$,*

1. *if* $\Sigma;\Gamma \vdash M : A$ *and* $\Sigma;\Gamma \vdash M : B$, *then* $[\![A]\!] = [\![B]\!]$, *and*
2. *if* $\Sigma;\Gamma \vdash S \triangleright \Delta_1$ *and* $\Sigma;\Gamma \vdash S \triangleright \Delta_2$, *then* $[\![\Delta_1]\!] = [\![\Delta_2]\!]$.

Proof. We only show the first case. The second case is proved by structural induction on Δ_1. By type uniqueness (Theorem 4.2), we have $A \equiv_{\lambda\Pi_\mathcal{L}} B$, and by sort soundness (Theorem 4.1), $A = B = Kind$ or ($\Sigma;\Gamma \vdash A : s_1$, $\Sigma;\Gamma \vdash B : s_2$, and $s_1, s_2 \in \{Kind, Type\}$). The first case is trivial. For the second one, we use soundness of $\lambda\Pi_\mathcal{L}$ (Theorem 4.5) to conclude that A and B are convertible via a path of well-typed terms. Hence, it suffices to prove that for any well-typed term N_1, if $N_1 \xrightarrow{\beta\Pi_\mathcal{L}} N_2$, then $[\![N_1]\!] = [\![N_2]\!]$. We prove this by induction on the depth of the $\beta\Pi_\mathcal{L}$-redex reduced in N_1. The only interesting case is (VarCons), i.e., $\underline{1}[M_1 \cdot_{A_1} S] \longrightarrow M_1$. We show that $[\![\underline{1}[M_1 \cdot_{A_1} S]]\!] = [\![M_1]\!]$.

- From Def. 5.11, $[\![\underline{1}[M_1 \cdot_{A_1} S]]\!] = [\![\underline{1}]\!] = \mathcal{SN}$.

- If $\underline{1}[M_1 \cdot_{A_1} S]$ is well-typed in $\Sigma;\Gamma$, then by inversion of rule (Cons), we have $\Sigma;\Gamma \vdash M_1 : A_1[S]$ and $\Sigma;\Gamma \vdash A_1[S] : \mathit{Type}$. Therefore, by Lemma 5.17, $[\![M_1]\!] = \mathcal{SN}$.

So, $[\![\underline{1}[M_1 \cdot_{A_1} S]]\!] = [\![M_1]\!] = \mathcal{SN}$. □

LEMMA 5.19. *Let $A_1 \in \mathcal{SN}$, and $M, A_2, B \in \mathcal{NF}$, if for all $N \in [\![A_2]\!]$, $(M[N \cdot_{A_1} \uparrow^0])\downarrow_{\Pi_\mathcal{L}} \subseteq [\![B]\!]$, then $\lambda_{A_1}.M \in [\![A_2]\!] \to [\![B]\!]$.*

Proof. Let $N \in [\![A_2]\!]$. We want to show $(\lambda_{A_1}.M\ N) \in [\![B]\!]$. Since $(\lambda_{A_1}.M\ N) \in \mathcal{NT}$ and $[\![B]\!] \subseteq \mathbf{SAT}$, it suffices to prove that if $(\lambda_{A_1}.M\ N) \xrightarrow{\beta\Pi_\mathcal{L}} M'$, then $M' \in [\![B]\!]$. By hypotheses, for all $N \in [\![A_2]\!]$, $(M[N \cdot_{A_1} \uparrow^0])\downarrow_{\Pi_\mathcal{L}} \subseteq [\![B]\!] \subseteq \mathcal{SN}$; in particular, $(M[\underline{1} \cdot_{A_1} \uparrow^0])\downarrow_{\Pi_\mathcal{L}} \subseteq \mathcal{SN}$. But, $M \in (M[\underline{1} \cdot_{A_1} \uparrow^0])\downarrow_{\Pi_\mathcal{L}}$, and thus, $M \in \mathcal{SN}$. We also have $N \in [\![A_2]\!] \subseteq \mathcal{SN}$ and $A_1 \in \mathcal{SN}$. Thus, we can reason by induction on $\nu(M) + \nu(N) + \nu(A_1)$. In one step $(\lambda_{A_1}.M\ N)$ $\beta\Pi_\mathcal{L}$-reduces to:

- $(M[N \cdot_{A_1} \uparrow^0])\downarrow_{\Pi_\mathcal{L}}$. By hypothesis, $(M[N \cdot_{A_1} \uparrow^0])\downarrow_{\Pi_\mathcal{L}} \subseteq [\![B]\!]$.
- $(\lambda_{A_1}.M\ N_1)$, with $N \xrightarrow{\beta\Pi_\mathcal{L}} N_1$. By Def. 5.4(2), $N_1 \in [\![A_2]\!]$, then by hypothesis, $(M[N_1 \cdot_{A_1} \uparrow^0])\downarrow_{\Pi_\mathcal{L}} \subseteq [\![B]\!]$. But also, $\nu(N_1) < \nu(N)$, thus, by induction hypothesis, $(\lambda_{A_1}.M\ N_1) \in [\![B]\!]$.
- $(\lambda_A.M\ N)$, with $A_1 \xrightarrow{\beta\Pi_\mathcal{L}} A$. But $A \in \mathcal{SN}$, since $A_1 \in \mathcal{SN}$, therefore, for any $M_1 \in (M[N \cdot_A \uparrow^0])\downarrow_{\Pi_\mathcal{L}}$, $\aleph(M_1) \subseteq \mathcal{SN}$. We have, $|(M[N \cdot_{A_1} \uparrow^0])\downarrow_{\Pi_\mathcal{L}}| = |(M[N \cdot_A \uparrow^0])\downarrow_{\Pi_\mathcal{L}}|$[4]. By Def. 5.4(4), $(M[N \cdot_A \uparrow^0])\downarrow_{\Pi_\mathcal{L}} \subseteq [\![B]\!]$. But also $\nu(A) < \nu(A_1)$, thus, by induction hypothesis, $(\lambda_A.M\ N) \in [\![B]\!]$.
- $(\lambda_{A_1}.M_1\ N)$, with $M \xrightarrow{\beta\Pi_\mathcal{L}} M_1$. Using the properties of $\lambda\Pi_\mathcal{L}$ and $\lambda\Pi_\mathcal{L}^\square$, if $N_1 \in (M[N \cdot_{A_1} \uparrow^0])\downarrow_{\Pi_\mathcal{L}}$, then $N_1 \xrightarrow{\beta\Pi_\mathcal{L}} N_2$, where $|N_2| = |(M_1[N \cdot_{A_1} \uparrow^0])\downarrow_{\Pi_\mathcal{L}}|$. By hypothesis, $N_1 \in [\![B]\!]$, thus, by Def. 5.4(2), $N_2 \in [\![B]\!]$. Since M_1 and A_1 are in $\mathcal{SN}$, for any $M_2 \in (M_1[N \cdot_{A_1} \uparrow^0])\downarrow_{\Pi_\mathcal{L}}$, $\aleph(M_2) \subseteq \mathcal{SN}$. We obtain $(M_1[N \cdot_{A_1} \uparrow^0])\downarrow_{\Pi_\mathcal{L}} \subseteq [\![B]\!]$ by Def. 5.4(4). But also $\nu(M_1) < \nu(M)$, thus, by induction hypothesis, $(\lambda_{A_1}.M_1\ N) \in [\![B]\!]$.

In any case, $(\lambda_{A_1}.M\ N)$ reduces to a term in $[\![B]\!]$ and, therefore, $(\lambda_{A_1}.M\ N) \in [\![B]\!]$. □

We are ready to prove the key lemma, the soundness of $\models$ with respect to $\vdash$.

LEMMA 5.20 (Soundness of $\models$). *Let $M, S \in \mathcal{NF}$,*

1. *if $\Sigma;\Gamma \vdash M : A$, then $\Gamma \models M : A$, and*
2. *if $\Sigma;\Gamma \vdash S \rhd \Delta$, then $\Gamma \models S \rhd \Delta$.*

Proof. Let $T \in [\![\Gamma]\!]$. We proceed by simultaneous structural induction on M and S. We show the main cases. In the proof, $\Uparrow_A(S)$ is a shorthand for $\underline{1} \cdot_A (S \circ \uparrow^1)$.

- $M = X$ (X is a meta-variable). We show that $(X[T])\downarrow_{\Pi_\mathcal{L}} \subseteq [\![A]\!]$.
  There are two cases:
  - $T = \uparrow^0$. Therefore, $(X[T])\downarrow_{\Pi_\mathcal{L}} = \{X\}$. But also, X is a neutral $\beta\Pi_\mathcal{L}$-normal form. Hence by Corollary 5.5, $X \in [\![A]\!]$.
  - $T \neq \uparrow^0$. Therefore, $(X[T])\downarrow_{\Pi_\mathcal{L}} = \{X[T]\}$. By Lemma 5.14, $T \in \mathcal{SN}$. Hence by Lemma 5.6, $X[T] \in [\![A]\!]$.
- $M = \Pi_{A_1}.B_1$. We show that $(\Pi_{A_1}.B_1[T])\downarrow_{\Pi_\mathcal{L}} \subseteq [\![A]\!]$.
  By inversion of rule (Prod), $\Sigma;\Gamma \vdash A_1 : \mathit{Type}$ and $\Sigma; A_1.\Gamma \vdash B_1 : s$, $s \in \{\mathit{Kind}, \mathit{Type}\}$. Note that if $M_1 \in ((\Pi_{A_1}.B_1)[T])\downarrow_{\Pi_\mathcal{L}}$, then $M_1 = \Pi_{A_2}.B_2$, where $A_2 \in (A_1[T])\downarrow_{\Pi_\mathcal{L}}$ and $B_2 \in (B_1[\Uparrow_{A_1}(T)])\downarrow_{\Pi_\mathcal{L}}$. By induction hypothesis on A_1, $(A_1[T_1])\downarrow_{\Pi_\mathcal{L}} \subseteq [\![\mathit{Type}]\!] = \mathcal{SN}$ holds for all $T_1 \in [\![\Gamma]\!]$. Assuming $T_1 = T$, we conclude $A_2 \in \mathcal{SN}$, and assuming $T_1 = \uparrow^0$, we conclude $A_1 \in \mathcal{SN}$.
  Let $T_2 \in (\Uparrow_{A_1}(T))\downarrow_{\Pi_\mathcal{L}}$. We have $|B_2| = |(B_1[T_2])\downarrow_{\Pi_\mathcal{L}}|$ and $T_2 \in [\![A.\Gamma]\!]$. By induction hypothesis on B_1, $(B_1[T_2])\downarrow_{\Pi_\mathcal{L}} \subseteq [\![s]\!] = \mathcal{SN}$ holds. But, $\aleph(B_2) \subseteq \mathcal{SN}$. Hence by Def. 5.4(4), $B_2 \in [\![s]\!] = \mathcal{SN}$.

[4]Since the $\Pi_\mathcal{L}^\square$-calculus ($\Pi_\mathcal{L}$ without annotations of types in substitutions) is confluent (Lemma 3.10), we use the following property: for any $M_1, M_2 \in (M)\downarrow_{\Pi_\mathcal{L}}$, $|M_1| = |M_2|$.

Since A_2, B_2 are both in $\mathcal{SN}$, we have $\Pi_{A_2}.B_2 \in [\![A]\!]$ (Lemma 5.7).

- $M = \lambda_{A_1}.M_1$. We show that $(\lambda_{A_1}.M_1[T])\downarrow_{\Pi_\mathcal{L}} \subseteq [\![A]\!]$.
  By inversion of rule (Abs), $\Sigma;\Gamma \vdash A_1 : \mathit{Type}$, $\Sigma; A_1.\Gamma \vdash M_1 : B$ and $\Sigma;\Gamma \vdash \lambda_{A_1}.M_1 : \Pi_{A_1}.B$. By Lemma 5.18, $[\![A]\!] = [\![\Pi_{A_1}.B]\!] = [\![A_1]\!] \to [\![B]\!]$. Note that if $N \in ((\lambda_{A_1}.M_1)[T])\downarrow_{\Pi_\mathcal{L}}$, then $N = \lambda_{A_2}.M_2$, where $A_2 \in (A_1[T])\downarrow_{\Pi_\mathcal{L}}$ and $M_2 \in (M_1[\Uparrow_{A_1}(T)])\downarrow_{\Pi_\mathcal{L}}$. By induction hypothesis on A_1, $(A_1[T_1])\downarrow_{\Pi_\mathcal{L}} \subseteq [\![\mathit{Type}]\!] = \mathcal{SN}$ holds for all $T_1 \in [\![\Gamma]\!]$. Assuming $T_1 = T$, we conclude $A_2 \in \mathcal{SN}$, and assuming $T_1 =\uparrow^0$, we conclude $A_1 \in \mathcal{SN}$.
  Now we prove that $\lambda_{A_2}.M_2 \in [\![A_1]\!] \to [\![B]\!]$. From Lemma 5.19, it suffices to prove that for any $N_1 \in [\![A_1]\!]$, $(M_2[N_1 \cdot_{A_2} \uparrow^0])\downarrow_{\Pi_\mathcal{L}} \subseteq [\![B]\!]$. Let $N_2 \in (M_2[N_1 \cdot_{A_2} \uparrow^0])\downarrow_{\Pi_\mathcal{L}}$ and $T_2 \in (\Uparrow_{A_1}(T) \circ (N_1 \cdot_{A_2} \uparrow^0))\downarrow_{\Pi_\mathcal{L}}$. We verify that $|N_2| = |(M_1[T_2])\downarrow_{\Pi_\mathcal{L}}|$ and $T_2 \in [\![A_1.\Gamma]\!]$. Therefore, by induction hypothesis on M_1, $(M_1[T_2])\downarrow_{\Pi_\mathcal{L}} \subseteq [\![B]\!]$. But $\aleph(N_2) \subseteq \mathcal{SN}$, thus, $N_2 \in [\![B]\!]$ by Def. 5.4(4).

□

Now, we show that $\beta\Pi_\mathcal{L}$ is strongly normalizing.

LEMMA 5.21 (Strong normalization of $\beta\Pi_\mathcal{L}$). *Let M be a term in $\mathcal{NF}$ and S be a substitution in $\mathcal{NF}$.*

1. If $\Sigma;\Gamma \vdash M : A$, then $M \in \mathcal{SN}$, and

2. if $\Sigma;\Gamma \vdash S \rhd \Delta$, then $S \in \mathcal{SN}$.

Proof. By Def. 5.13, $\uparrow^0 \in [\![\Gamma]\!]$.

1. By Lemma 5.20, $M \in (M[\uparrow^0])\downarrow_{\Pi_\mathcal{L}} \subseteq [\![A]\!]$. By Corollary 5.12 and Def. 5.4(1), $[\![A]\!] \subseteq \mathcal{SN}$.
2. By Lemma 5.20, $S \in (S \circ \uparrow^0)\downarrow_{\Pi_\mathcal{L}} \subseteq [\![\Delta]\!]$, and by Lemma 5.14, $[\![\Delta]\!] \subseteq \mathcal{SN}$.

□

Finally, we prove weak normalization on well-typed $\lambda\Pi_\mathcal{L}$-expressions.

THEOREM 5.22 (Weak normalization). *Let M be a term in $\lambda\Pi_\mathcal{L}$ and S a substitution in $\lambda\Pi_\mathcal{L}$.*

1. If $\Sigma;\Gamma \vdash M : A$, then M is weakly normalizing, and

2. if $\Sigma;\Gamma \vdash S \rhd \Delta$, then S is weakly normalizing.

Therefore, M and S have $\lambda\Pi_\mathcal{L}$-normal forms.

Proof. By Lemma 2.1 there exist $M_1, S_1 \in \mathcal{NF}$ such that $M \xrightarrow{\Pi_\mathcal{L}^*} M_1$ and $S \xrightarrow{\Pi_\mathcal{L}^*} S_1$. The subject reduction theorem (Theorem 4.3) states that typing is preserved under reductions. Hence, $\Sigma;\Gamma \vdash M_1 : A$ and $\Sigma;\Gamma \vdash S_1 \rhd \Delta$. Therefore, by Lemma 5.21, M_1 and S_1 are both in $\mathcal{SN}$. Finally, note that $\beta\Pi_\mathcal{L}$-normal forms in $\mathcal{NF}$ are $\lambda\Pi_\mathcal{L}$-normal forms, too. □

5.2. Confluence. The Church-Rosser property states that if two well-typed expressions are convertible, then they are joinable. The confluence property states that all the reductions of a well-typed expression are joinable.

We need the following lemma coined in [44].

LEMMA 5.23. *Let x and y be $\lambda\Pi_\mathcal{L}$-normal forms such that $x \equiv_{\lambda\Pi_\mathcal{L}} y$. Then, $x = y$ if*

- *x is a term, $\Sigma;\Gamma_1 \vdash x : A$ and $\Sigma;\Gamma_2 \vdash y : B$, or*
- *x is a substitution, $\Sigma;\Gamma_1 \vdash x \rhd \Delta_1$, $\Sigma;\Gamma_2 \vdash y \rhd \Delta_2$, and $\Delta_1 \equiv_{\lambda\Pi_\mathcal{L}} \Delta_2$.*

Proof. By Lemma 3.2(3), $|x|$ and $|y|$ are $\lambda\Pi_\mathcal{L}^\square$-normal forms, and by Lemma 3.2(1), $|x| \equiv_{\lambda\Pi_\mathcal{L}^\square} |y|$. Since $\lambda\Pi_\mathcal{L}^\square$ is confluent (Theorem 3.7), $|x| = |y|$ holds. Finally, we proceed by structural induction on x. We use the fact that sub-terms of well-typed normal forms are well-typed normal forms. The only interesting case is $x = M[T]$. Since x is a $\lambda\Pi_\mathcal{L}$-normal form, only two cases are possible:

- $M = \underline{1}$ and $T =\uparrow^{n+1}$. This case is trivial, since by Def. 3.1, $\underline{1}[\uparrow^{n+1}] = |\underline{1}[\uparrow^{n+1}]|$. Therefore, $x = y$.
- $M = X$, where X is a meta-variable and $T \neq \uparrow^0$. By hypothesis, $y = X[T_1]$ where $|T| = |T_1|$. By Lemma 3.3, $T \equiv_{\lambda\Pi_\mathcal{L}} T_1$. Let Δ be the type of T and Δ_1 the type of T_1. By the inversion of rule

(Clos) applied to x and y, it holds that X is well-typed in both contexts Δ and Δ_1. By inversion of rule (Metavar), $\Delta \equiv_{\lambda\Pi_{\mathcal{L}}} \Delta_1$. Thus, by induction hypothesis, $T = T_1$, and thus, $x = y$.

□

The above property is not valid when $\Delta_1 \not\equiv_{\lambda\Pi_{\mathcal{L}}} \Delta_2$. Take, for example, the context

$$\Gamma = m{:}(T\ 0) \rightarrow nat.\ 0{:}nat.\ l{:}(\Pi n{:}nat.(T\ n)).\ T{:}nat \rightarrow Type.\ nat{:}Type$$

and the two substitutions

$$S_1 = [y := (l\ 0) \cdot_{(T\ x)} x := 0 \cdot_{nat} \uparrow^0]$$

and

$$S_2 = [y := (l\ 0) \cdot_{(T\ 0)} x := 0 \cdot_{nat} \uparrow^0].$$

By Lemma 3.3, $S_1 \equiv_{\lambda\Pi_{\mathcal{L}}} S_2$. Also,

$$\Gamma \vdash S_1 \triangleright y{:}(T\ x).\ x{:}nat.\ \Gamma$$

and

$$\Gamma \vdash S_2 \triangleright y{:}(T\ 0).\ x{:}nat.\ \Gamma.$$

In this case, the well-typed substitutions S_1 and S_2 are $\equiv_{\lambda\Pi_{\mathcal{L}}}$-convertible, but they are not identical.

THEOREM 5.24 (Church-Rosser). *Let x and y be such that $x \equiv_{\lambda\Pi_{\mathcal{L}}} y$. Then, x and y are $\lambda\Pi_{\mathcal{L}}$-joinable, i.e., there exists w such that $x \xrightarrow{\lambda\Pi_{\mathcal{L}}^*} w$ and $y \xrightarrow{\lambda\Pi_{\mathcal{L}}^*} w$, if*

1. *x is a term, $\Sigma;\Gamma_1 \vdash x : A$ and $\Sigma;\Gamma_2 \vdash y : B$, or*
2. *x is a substitution, $\Sigma;\Gamma_1 \vdash x \triangleright \Delta_1$, $\Sigma;\Gamma_2 \vdash y \triangleright \Delta_2$, and $\Delta_1 \equiv_{\lambda\Pi_{\mathcal{L}}} \Delta_2$.*

Proof. By weak normalization theorem (Theorem 5.22), there exists $\lambda\Pi_{\mathcal{L}}$-normal forms x' and y' such that $x \xrightarrow{\lambda\Pi_{\mathcal{L}}^*} x'$ and $y \xrightarrow{\lambda\Pi_{\mathcal{L}}^*} y'$. It suffices to show that $x' = y'$, which is a consequence of subject reduction theorem (Theorem 4.3) and Lemma 5.23. □

Confluence of $\lambda\Pi_{\mathcal{L}}$ is a consequence of the Church-Rosser property (Theorem 5.24) and subject reduction (Theorem 4.3).

COROLLARY 5.25 (Confluence). *Let x be an arbitrary well-typed expression. If $x \xrightarrow{\lambda\Pi_{\mathcal{L}}^*} y$ and $x \xrightarrow{\lambda\Pi_{\mathcal{L}}^*} z$ for some y, z, then there exists w such that $y \xrightarrow{\lambda\Pi_{\mathcal{L}}^*} w$ and $z \xrightarrow{\lambda\Pi_{\mathcal{L}}^*} w$.*

Since $\lambda\Pi_{\mathcal{L}}$ enjoys both Church-Rosser and weak normalization, we have that $\lambda\Pi_{\mathcal{L}}$-normal forms on well-typed terms always exist and they are unique. Thus, the equivalence on well-typed expressions is decidable.

COROLLARY 5.26 (Decidability). *The equivalence $x \equiv_{\lambda\Pi_{\mathcal{L}}} y$ is decidable if*

- *x is a term, $\Sigma;\Gamma_1 \vdash x : A$ and $\Sigma;\Gamma_2 \vdash y : B$, or*
- *x is a substitution, $\Sigma;\Gamma_1 \vdash x \triangleright \Delta$, $\Sigma;\Gamma_2 \vdash y \triangleright \Delta$.*

6. Related Work and Conclusion. Explicit substitutions and the `let-in` constructor of functional ML-style programming languages have similar characteristics. In both mechanisms the application of a substitution to a term can be delayed. For example, `let` $x := 0$ `in` $\lambda y{:}A.x$ will be unfolded to $\lambda y{:}A.0$, in the same way that $(\lambda y{:}A.x)[x := 0]$ reduces to $\lambda y{:}A.0$. In their simply-typed versions, explicit substitutions and `let-in` constructors act in the same way. However, in dependent-type systems, the relationship between both mechanisms is not immediate.

To illustrate this, let us take the typing rule for closures —explicit applications of substitutions to terms— in a dependent-type system:

$$\frac{\Gamma \vdash S \triangleright \Delta \quad \Delta \vdash M : A \quad \ldots}{\Gamma \vdash M[S] : A[S]} \;(\text{Clos}_\Pi).$$

Consider the context

$$\Gamma = m{:}(T\ 0) \rightarrow nat.\ 0{:}nat.\ l{:}(\Pi n{:}nat.(T\ n)).\ T{:}nat \rightarrow Type.\ nat{:}Type.$$

Using the above typing rule, the term $(m\ (l\ x))[x := 0]$ is ill-typed. This is because the information that the variable x will be substituted by 0 in $(m\ (l\ x))$ is not taken into account by rule (Clos$_\Pi$). Therefore, the type of $(l\ x)$ is $(T\ x)$, but not $(T\ 0)$ as expected by m. On the other hand, the same term can be written using the `let-in` notation as: `let` $x := 0$ `in` $(m\ (l\ x))$. This term is well-typed because x has the value 0 in $(m\ (l\ x))$, and thus `let` $x := 0$ `in` $(m\ (l\ x))$ is going to be typed as $(m\ (l\ 0))$.

The unfolding of definitions before typing is not sufficient when we admit meta-variables. The reason is that substitutions and meta-variables may appear in normal forms. In this case, we cannot avoid having a (Clos$_\Pi$)'s like rule. The approach we have taken is to consider explicit substitutions different from the `let-in` mechanism. The explicit substitution technique allows substitutions to be part of the formal language by means of special constructors and reduction rules. In this way, the term $(m\ (l\ x))[x := 0]$ is ill-typed, just as the term $(\lambda x{:}nat.(m\ (l\ x))\ 0)$ is. The `let-in` structure has a more complex behavior. It provides a mechanism for definitions in the language. Formal presentations of type systems with definitions are given in [41, 3].

Some type theories extended with explicit substitutions have been proposed: The Simple Type Theory [1, 27, 8, 21, 6], the Second-Order Type Theory [1], the Martin Löf Type Theory [43], the Calculus of Constructions [39], and Pure Type Systems [2]. Except for the simply-typed version of $\lambda\sigma$ in [8], neither of them considers terms with meta-variables as first-class objects.

Our main contribution is the complete meta-theoretical development of a dependent-type system with explicit substitutions which handles explicitly open expressions (i.e., expressions with meta-variables). The system enjoys the usual typing properties: type uniqueness, subject reduction, weak normalization, and confluence. Applications of such a calculus are frameworks for the representation of incomplete proofs, and first-order settings for higher-order unification problems.

In this paper, we have presented the $\lambda\Pi$-theory. Although full polymorphism or inductive definitions are not considered in this theory, the main difficulties, due to the mutual dependence between terms and types, already arise in $\lambda\Pi$. Other theories, such as the Calculus of Constructions, can be considered as the logical framework for $\lambda\Pi_{\mathcal{L}}$ [34]. Note also, that $\lambda\Pi_{\mathcal{L}}$ does not handle the η-rule. Extensional versions of explicit substitution calculi have been studied for ground terms [24]. However, work is necessary to understand the interaction with dependent types and meta-variables.

Acknowledgments. A major part of this research was done while the author was a research assistant in the Coq Project at INRIA-Rocquencourt. Many persons have contributed to this work with useful remarks and suggestions, in particular Gilles Dowek, Delia Kesner, and Nikolaj Bjørner. The author is very grateful to them.

REFERENCES

[1] M. ABADI, L. CARDELLI, P.-L. CURIEN, AND J.-J. LÉVY, *Explicit substitution*, Journal of Functional Programming, 1 (1991), pp. 375–416.

[2] R. BLOO, *Preservation of Termination for Explicit Substitution*, Ph.D. thesis, Eindhoven University of Technology, 1997.

[3] R. BLOO, F. KAMAREDDINE, AND R. NEDERPELT, *The Barendregt cube with definitions and generalised reduction*, Information and Computation, 126 (1996), pp. 123–143.

[4] R. BLOO AND K. H. ROSE, *Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection*, in Proceedings of CSN-95: Computer Science in the Netherlands, Nov. 1995.

[5] D. BRIAUD, *Higher order unification as a typed narrowing*, CRIN report 96-R-112, 1996.

[6] R. D. COSMO AND D. KESNER, *Strong normalization of explicit substitutions via cut elimination in proof nets (extended abstract)*, in Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, 29 June–2 July 1997, IEEE Computer Society Press, pp. 35–46.

[7] P.-L. CURIEN, T. HARDIN, AND J.-J. LÉVY, *Confluence properties of weak and strong calculi of explicit substitutions*, Journal of the ACM, 43 (1996), pp. 362–397.

[8] G. DOWEK, T. HARDIN, AND C. KIRCHNER, *Higher-order unification via explicit substitutions (extended abstract)*, in Proceedings of the Tenth Annual IEEE Symposium on Logic in Computer Science, San Diego, California, 26–29 June 1995, IEEE Computer Society Press, pp. 366–374.

[9] G. DOWEK, T. HARDIN, C. KIRCHNER, AND F. PFENNING, *Unification via explicit substitutions: The case of higher-order patterns*, in Proceedings of the Joint International Conference and Symposium on Logic Programming, M. Maher, ed., Bonn, Germany, Sept. 1996, MIT Press.

[10] M. C. F. FERREIRA, D. KESNER, AND L. PUEL, *Lambda-calculi with explicit substitutions and composition which preserve beta-strong normalization*, in Algebraic and Logic Programming, Fifth International Conference, ALP'96, M. Hanus and M. Rodríguez-Artalejo, eds., Vol. 1139 of LNCS, Aachen, Germany, 25–27 Sept. 1996, Springer, pp. 284–298.

[11] H. GEUVERS, *The Church-Rosser property for $\beta\eta$-reduction in typed λ-calculi*, in Proceedings of the Seventh Annual IEEE Symposium on Logic in Computer Science, Santa Cruz, California, 22–25 June 1992, IEEE Computer Society Press, pp. 453–460.

[12] ——, *A short and flexible proof of strong normalization for the calculus of constructions*, in Selected Papers 2nd Intl. Workshop on Types for Proofs and Programs, TYPES'94, Båstad, 6–10 June 1994, P. Dybjer, B. Nordström, and J. Smith, eds., Vol. 996 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1995, pp. 14–38.

[13] H. GEUVERS AND R. BLOO, *Counter-example for subject reduction in calculi of explicit substitutions with dependent types*, personal communication, 1997.

[14] H. GEUVERS AND B. WERNER, *On the Church-Rosser property for expressive type systems and its consequences for their metatheoretic study*, in Proceedings of the Ninth Annual IEEE Symposium on Logic in Computer Science, Paris, 4–7 July 1994, IEEE Computer Society Press, pp. 320–329.

[15] J.-Y. GIRARD, *Interprétation Fonctionelle et Élimination des Compures de l'Arithmétic d'Ordre Supérieur*, thèse de doctorat, Université Paris VII, 1972.

[16] J.-Y. GIRARD, P. TAYLOR, AND Y. LAFONT, *Proof and Types*, Cambridge University Press, 1989.

[17] J. GOUBAULT-LARRECQ, *A proof of weak termination of typed lambda-calculi*, Lecture Notes in Com-

puter Science, 1512 (1998), pp. 134–153.

[18] R. Harper, F. Honsell, and G. Plotkin, *A framework for defining logics*, Journal of the Association for Computing Machinery, 40 (1993), pp. 143–184.

[19] G. Huet, *Confluent reductions: Abstract properties and applications to term rewriting systems*, J.A.C.M., 27 (1980).

[20] F. Kamareddine and A. Ríos, *A lambda-calculus à la De Bruijn with explicit substitutions*, LNCS, 982 (1995), pp. 45–62.

[21] ——, *The λ_f-calculus: Its typed and its extended versions*, personal communication, June 1995.

[22] D. Kapur, P. Narendran, and F. Otto, *On ground-confluence of term rewriting systems*, Information and Computation, 86 (1990), pp. 14–31.

[23] D. Kapur and H. Zhang, *RRL: A rewrite rule laboratory-user's manual*, Tech. Report 89-03, Department of Computer Science, University of Iowa, 1989.

[24] D. Kesner, *Confluence properties of extensional and non-extensional λ-calculi with explicit substitutions (extended abstract)*, in Proceedings of the Seventh International Conference on Rewriting Techniques and Applications (RTA-96), H. Ganzinger, ed., Vol. 1103 of LNCS, New Brunswick, New Jersey, 1996, Springer-Verlag, pp. 184–199.

[25] C. Kirchner and C. Ringeissen, *Higher order equational unification via explicit substitutions*, in Proceedings of the International Conference PLILP/ALP/HOA'97, Vol. 1298 of LNCS, Southampton, Sept. 1997, Springer.

[26] J.-W. Klop, *Combinatory reduction systems*, Mathematical Center Tracts, (1980).

[27] P. Lescanne, *From $\lambda\sigma$ to $\lambda\upsilon$ a journey through calculi of explicit substitutions*, in Proceedings of the 21st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Jan. 1994, pp. 60–69.

[28] P. Lescanne and J. Rouyer-Degli, *Explicit substitutions with de Bruijn's levels*, in Proceedings of the International Conference on Rewriting Techniques and Applications (RTA-95), J. Hsiang, ed., Vol. 914 of LNCS, Chapel Hill, North Carolina, 1995, Springer-Verlag, pp. 294–308.

[29] L. Magnusson, *The Implementation of ALF—A Proof Editor Based on Martin-Löf's Monomorphic Type Theory with Explicit Substitution*, Ph.D. thesis, Chalmers University of Technology and Göteborg University, Jan. 1995.

[30] P. A. Mellies, *Typed lambda-calculi with explicit substitutions may not terminate*, LNCS, 902 (1995), pp. 328–338.

[31] C. Muñoz, *Confluence and preservation of strong normalisation in an explicit substitutions calculus (extended abstract)*, in Proceedings of the Eleventh Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, July 1996, IEEE Computer Society Press, pp. 440–447.

[32] ——, *Proof representation in type theory: State of the art*, in Proceedings of the XXII Latinamerican Conference of Informatics CLEI Panel 96, Santafé de Bogotá, June 1996.

[33] ——, *A left-linear variant of $\lambda\sigma$*, in Proc. International Conference PLILP/ALP/HOA'97, Vol. 1298 of LNCS, Southampton, Sept. 1997, Springer, pp. 224–234.

[34] ——, *Un calcul de substitutions pour la représentation de preuves partielles en théorie de types*, thèse de doctorat, Université Paris VII, 1997. English version available as INRIA research report RR-3309.

[35] G. Nadathur, *The (SCons) rule*, personal communication, 1996.

[36] R. P. Nederpelt, *Strong normalization in a typed lambda calculus with lambda structured types*, Ph.D. thesis, Technical University Eindhoven, Eindhoven, 1973.

[37] B. PAGANO, *Confluent extensions of* $\lambda_{\Uparrow}$, personal communication, 1996.

[38] A. RÍOS, *Contributions à l'étude de λ-calculs avec des substitutions explicites*, thèse de doctorat, Université Paris VII, 1993.

[39] E. RITTER, *Categorical abstract machines for higher-order lambda calculi*, Theoretical Computer Science, 136 (1994), pp. 125–162.

[40] M. SCHMIDT-SCHAUSS, *Computational aspects of an order-sorted logic with term declarations*, Vol. 395 of LNCS and LNAI, Springer-Verlag, New York, 1989.

[41] P. SEVERI, *Normalisation in LAMBDA CALCULUS and its relation to type inference*, Ph.D. thesis, Eindhoven University of Technology, 1996.

[42] W. W. TAIT, *Intentional interpretation of functionals of finite type i*, Journal of Symbolic Logic, 32 (1967).

[43] A. TASISTRO, *Formulation of Martin-Löf's theory of types with explicit substitutions*, tech. report, Chalmers University of Technology, University of Göteborg, Göteborg, May 1993.

[44] B. WERNER, *Une Théorie des Constructions Inductives*, thèse de doctorat, Université Paris VII, 1994.

[45] H. YOKOUCHI AND T. HIKITA, *A rewriting system for categorical combinators with multiple arguments*, SIAM Journal on Computing, 19 (1990), pp. 78–97.

[46] H. ZANTEMA, *Termination of term rewriting by semantic labelling*, Fundamenta Informaticae, 24 (1995), pp. 89–105.

[47] ——, *Termination of* ϕ *and* Π_ϕ *by semantic labeling*, personal communication, 1996.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY*(Leave blank)*	**2. REPORT DATE** November 1999	**3. REPORT TYPE AND DATES COVERED** Contractor Report

4. TITLE AND SUBTITLE Dependent types and explicit substitutions	**5. FUNDING NUMBERS** C NAS1-97046 WU 505-90-52-01
6. AUTHOR(S) César Mũnoz	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23681-2199	**8. PERFORMING ORGANIZATION REPORT NUMBER** ICASE Report No. 99-43
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Langley Research Center Hampton, VA 23681-2199	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER** NASA/CR-1999-209722 ICASE Report No. 99-43

11. SUPPLEMENTARY NOTES
Langley Technical Monitor: Dennis M. Bushnell
Final Report
Submitted to Mathematical Stuctures in Computer Science.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified–Unlimited Subject Category 60, 61 Distribution: Nonstandard Availability: NASA-CASI (301) 621-0390	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 words)*
We present a dependent-type system for a λ-calculus with explicit substitutions. In this system, meta-variables, as well as substitutions, are first-class objects. We show that the system enjoys properties like type uniqueness, subject reduction, soundness, confluence and weak normalization.

14. SUBJECT TERMS explicit substitutions, dependent types, lambda-calculus	**15. NUMBER OF PAGES** 31
	16. PRICE CODE A03

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**